COLUMN RECOGNITON AND DEFECTS/DAMAGE PROPERTIES RETRIEVAL FOR RAPID INFRASTRUCTURE ASSESSMENT AND REHABILITATION USING MACHINE VISION

A Dissertation
Presented to
The Academic Faculty

By

Zhenhua Zhu

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
School of Civil and Environmental Engineering

Georgia Institute of Technology

August, 2011

COLUMN RECOGNITON AND DEFECTS/DAMAGE PROPERTIES RETRIEVAL FOR RAPID INFRASTRUCTURE ASSESSMENT AND REHABILITATION USING MACHINE VISION

Approved by:

Dr. Ioannis Brilakis, Advisor
School of Civil and Environmental Engineering
Georgia Institute of Technology

Dr. Reginald DesRoches, Co-Advisor
School of Civil and Environmental Engineering
Georgia Institute of Technology

Dr. Daniel Castro-Lacouture
School of Building Construction
Georgia Institute of Technology

Dr. Jochen Teizer
School of Civil and Environmental Engineering
Georgia Institute of Technology

Dr. Silvio Savarese
Department of Electrical and Computer Engineering
University of Michigan – Ann Arbor

Date Approved: May 16, 2011

ACKNOWLEDGEMENTS

This dissertation completes one important segment in my life. It is hard to accomplish and extremely far from what I expected before. I would like to thank all people who have helped and inspired me to make this dissertation possible.

First and foremost, I offer my sincere gratitude to my advisor, Dr. Ioannis Brilakis, and co-advisor, Dr. Reginald DesRoches. They have supported me during my doctoral studies with their patience and knowledge whilst allowing me the room to work in my own way. Their perpetual enthusiasms in research have motivated me, and as a result, my research life at Georgia Tech is rewarding.

I am delighted to have Dr. Daniel Castro-Lacouture, Dr. Silvio Savarese, and Dr. Jochen Teizer become my dissertation committee members. Their expertise and experience broaden my perspectives and nourish my intellectual maturity. I gratefully acknowledge them for their valuable guidance and comments.

Many thanks go to my current and former lab mates in the Construction Information Technology Laboratory. They make the lab a convivial place to work. They are: Dr. Fei Dai, Mr. Man-Woo Park, Ms. Gauri Jog, Mr. Habib Fathi, Mr. Abbas Rashidi, Ms. Sara Roberts, Ms. Stephanie German, Dr. Christian Koch, Ms. Atefe Makhmalbaf, Ms. Aswathy Sivaram, Mr. Abhishek Gupta, Mr. Paavan Vasudev, Mr. Francisco Cordova, Mr. Changli Wang, and Mr. Chieh-Cheng Huang. In particular, I would like to thank Ms. Stephanie German. It is a pleasure to collaborate with you in designing experiments, and writing research proposals and papers. I would also like to thank Dr. Christian Koch and Ms. Gauri Jog. You inspired my research and life.

Where would I be without my family? No words can express my appreciation to my parents. My Father, Jianmin Zhu, is a typical Chinese father, who works industriously to provide the best possible environment for me to grow up without any complaints. My Mother, Guoqin Cai, is the one who sincerely raises me with her ever-lasting love and constant support throughout my life. Both of them showed me the joy of intellectual pursuit ever since I was a child.

I will not forget my doctoral study years at Georgia Tech. Again, I would like thank everybody who is important to the successful realization of this dissertation. Also, I like to thank the National Science Foundation for its indirect and direct financial support on this research under grant #0625643, #0948415, and #1000700.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

No matter how inspection techniques have been advanced, manual visual inspection is currently still the first and fundamental step in assessing civil infrastructure. If no sign of deterioration has been spotted in manual inspection, any future inspection actions is not necessary to take. However, manual inspection has been identified with several limitations including the qualitative nature of inspection results, the time-consuming inspection process, and the heavy reliance on inspectors' and/or engineers' experience. In order to overcome these limitations, automated visual inspection systems have been proposed to enhance and/or replicate the manual inspection process. A number of image processing methods have been developed in detecting defects (i.e. coating rusts) and damage (i.e. cracks) on civil infrastructure. Their effectiveness has been verified in inspecting structures, such as bridges, underground pipes, and tunnels.

Although existing methods are effective in finding defects and damage from digital images, missing two important links limits their application for rapid infrastructure assessment and rehabilitation. The first link is the correlation between the defects/damage and the structural members that the defects/damage lie on. The second link is the relationship between the defects/damage and their impacts on the structural members.

The purpose of this research is to investigate the way of establishing these two links. It is focused on the retrieval of critical structural members and defects/damage information from images/videos, and then the utilization of this information for automated and rapid assessment and rehabilitation of civil infrastructure. Specifically, a combination of techniques from the areas of visual pattern recognition, digital filtering, and machine

vision have been used in order to develop a set of methods for concrete column recognition, crack properties retrieval, and air pockets and discoloration detection and evaluation. The methods proposed in this research were implemented in a Microsoft Visual Studio environment, and tested on the real images/videos of concrete structures inflicted with cracks, air pockets and discoloration. The test results indicated that the methods could automatically recognize concrete columns, correctly measure the properties of the cracks in a crack map, and accurately evaluate the impacts of air pockets and discoloration on the visual quality of concrete surfaces.

CHAPTER 1
INTORUDCTION

This research seeks to demonstrate that the techniques in the area of visual pattern recognition, digital filtering, and machine vision can be used to retrieve critical structural members and defects/damage information from images and videos. This information can be further used for automated and rapid assessment and rehabilitation of civil infrastructure in both routine and post-earthquake scenarios. The following sections in this chapter introduce the research motivation, objectives, methodology, contributions, and the organization of this dissertation.

1.1 Background and Motivation

The National Academy of Engineering recently listed "Restoring and Improving Urban Infrastructure" as one of the Grand Challenges of Engineering in the 21st century (NAE, 2008). On the one hand, portions of aging and deteriorating civil infrastructure require significant maintenance, expansion, and modernization. The latest report card issued by the American Society of Civil Engineers (ASCE) indicated that the average grade of America's infrastructure was "D", and an investment of $2.2 trillion over five years was needed to bring the nation's infrastructure up to a good condition (ASCE, 2009). Take highway bridges, which was graded "C" in the report, for an example. According to the data from the Bureau of Transportation Statistics up to 2008, there were more than 600,000 highway bridges in the United States, around 25% of which were rated as either structurally deficient or functionally obsolete or both (BTS, 2008). At the current spending level (roughly $10.5 billion per year), it was estimated that the annual

gap of 1.9 billion dollars was required to reduce this percentage to near zero by 2024 (Kirk and Mallett, 2007). On the other hand, the increased risks of natural and manmade hazards and disasters have placed significant impacts on the urban built environment. In the 2010 Haiti earthquake, 250,000 residences and 30,000 commercial buildings collapsed and were severely damaged (Renois, 2010) (Figure 1). Similarly, the 2010 Chile earthquake destroyed more than 500,000 homes (MarketWatch, 2010).

Figure 1: Damaged structures after the 2010 Haiti earthquake

One critical component behind restoring and improving urban infrastructure is infrastructure inspection and assessment. For example, each of highway bridges in the United States requires inspection at regular intervals on a routine basis to determine their physical and functional conditions and ensure that they still satisfy present service requirements. The intervals are usually not exceeding two years with few exceptions

(AASHTO, 2001). In an emergency scenario, the safety of entry into damaged buildings needs to be immediately evaluated after an earthquake occurs, so that emergency responders, such as fire fighters, can go into the buildings to perform life search and rescue tasks. Also, buildings need to be inspected, assessed, and classified for residents as 1) posing and imminent threat to life-safety (red-tag), 2) posing some risk from damage but not imminent threat to life-safety (yellow-tag), or 3) safe for entry and occupancy as earthquake damage has not significantly affected the safety of the building (green-tag).

Many techniques, such as stress wave, radiation, and infrared thermograph, have been developed to facilitate the procedures of civil infrastructure inspection and assessment. Stress wave methods utilize the propagation of a wave (e.g. an ultrasonic wave or other kinds of waves introduced by sudden impacts at surface points) to test concrete structures. Radiation methods assess the density of fresh or hardened concrete on structural members with high-energy electromagnetic radioactive materials. Infrared thermography methods sense the emission of thermal or infrared radiation from a concrete member surface and then measure the variations of the radiance to detect subsurface anomalies. Researchers have applied these techniques in assessing the delaminations in bridge decks (Gucunski, and Consolazio, 2006), detecting the existence of internal defects in concrete structures (Kamoi et al. 2004), and evaluating the fatigue of steel welded joints (Galietti and Palumbo, 2010).

Although these techniques can provide detailed and quantitative data on infrastructure conditions, the use of these techniques requires expensive equipment and a significant amount of time and skilled operators for equipment setup and data interpretation. For example, the cost of a quantitative ground-penetrating radar (GPR) survey on the I-35W

Bridge in Minneapolis was estimated around $40,000 in 2006 (USA Today, 2008). Therefore, these advanced techniques are limited to use either routinely or in a post-disaster scenario, considering the enormous amount of existing civil infrastructure in the United States (600,000 bridges, 4,000,000 miles of public roadway, and 1,000,000 miles of water mains) and the huge number of structures necessary to inspect and assess after a moderate earthquake.

Today, manual visual inspection supplemented with other qualitative methods like judging the sounds produced by dragging chains or hammering is still the first and primary step in infrastructure inspection and assessment, and has been widely used in practice. The inspection is usually performed by qualified/certified inspectors or engineers with a wealth of knowledge (ACI 228.2R-98, 2005). Inspection results are mainly based on the inspectors' observations and visual assessments. The results represent the condition information of structural members (e.g. columns, beams, and decks/slabs). This information can be used to help state or city agencies make system-wide decisions in allocating limited construction maintenance and rehabilitating resources. Also, the results can be used to identify the necessity of performing further inspections. If manual inspection shows no sign of deterioration, any further actions are not necessary to take and the inspection for the next period can then be scheduled (Sitar, 2005). For example, in the case of assessing existing nuclear safety-related concrete structures, ACI 349.3R (2005) clearly points out that the structures are generally acceptable without any other assessments, when their surfaces satisfy the diameter of air pockets less than 20 mm.

However, there are several limitations associated with manual visual inspection. First, the results from manual inspection are subjective and not always reliable (Moore et al. 2001; Phares et al. 2004). Second, the process of manual inspection is time-consuming (Bartel, 2001; Jauregui and White, 2003; Koglin, 2007). When the lack of available inspectors is combined with the large volume of inspection work after an earthquake, completing the required inspection always takes weeks or even months (Chock ,2007). Third, a number of safety risks are associated with the inspections since inspectors sometimes work at high heights or in heavy traffic zones (NJDOT, 2009). Also, the requirement of experienced inspectors in bridge inspection poses a challenge for the construction industry, which is now facing the pressing shortage of experienced and highly trained inspection personnel (TPFP, 2009).

The limitations have huge impacts on society and nation. The subjective nature of manual visual inspection has been reported not sufficiently reliable for optimal civil infrastructure management (FHWA, 2001). For example, the I35W Bridge in Minneapolis was scheduled to be replaced in 2020, but it collapsed in 2007 (FOX News, 2007). The large amount of inspection hours and the requirement of skilled inspectors have also proven costly over time. In Tennessee, the cost of bridge inspection was $7.6 million in the fiscal year of 2006 – 2007. Considering the Tennessee Department of Transportation lost $22 million in Federal bridge funds in the last two years, some bridge maintenance projects have to be pushed back (Wasserman, 2007). After an earthquake, the affected population is homeless and/or jobless. The long elapsed time spent in inspecting and assessing damaged commercial and residential buildings has adverse

effects on the recovery from economic and social disruptions induced by the earthquake (Kamat and El-Tawil, 2007).

In order to overcome these limitations, automated visual inspection systems have been proposed to enhance or even replicate the current practice of manual visual inspection. The input of the systems is typically a set of high-resolution images captured by a Charged Couple Device (CCD) camera. The high-resolution images provide the detailed defects/damage information on the surfaces of structural members. The defect/damage detection methods in the systems try to automatically retrieve this information by distinguishing the defects/damage areas from the surface background with pattern recognition, digial filtering, and machine vision techniques, such as wavelet transforms, Fourier transforms, thresholding, edge detection, and/or region-based segmentation. The effectiveness of these methods has been verified in inspecting structures like concrete bridges (Abdel-Qader et al. 2006; Abdel-Qader et al. 2006), underground pipes (Sinha and Fieguth, 2006; Guo et al. 2008), and tunnels (Yu et al. 2006). This validates the ability of these methods in detecting defects and damage for addressing civil infrastructure related assessment and rehabilitation problems, even when well-light conditions are not available (e.g. underground pipes).

Although existing defects/damage detection methods are effective in finding defects and damage from digital images, missing two important links limits the application of the systems in automated rapid infrastructure assessment and rehabilitation (Figure 2). The first link is the connection between the defects/damage and the structural members that the defects/damage lie on. The detected defects/damage is of little value for infrastructure assessment and rehabilitation unless they are correlated with the members on which they

lie. For example, a diagonal crack on a column and a diagonal crack on a beam indicate two completely different types of damage on a structure. The second link is the relationship between the defects/damage and their impacts on infrastructure members. Again, take cracks on a concrete column for an example. A diagonal crack versus a horizontal (flexural) crack indicates a completely different type of damage on the column. The main purpose of this research is to investigate the way of establishing these two important links from images and videos.

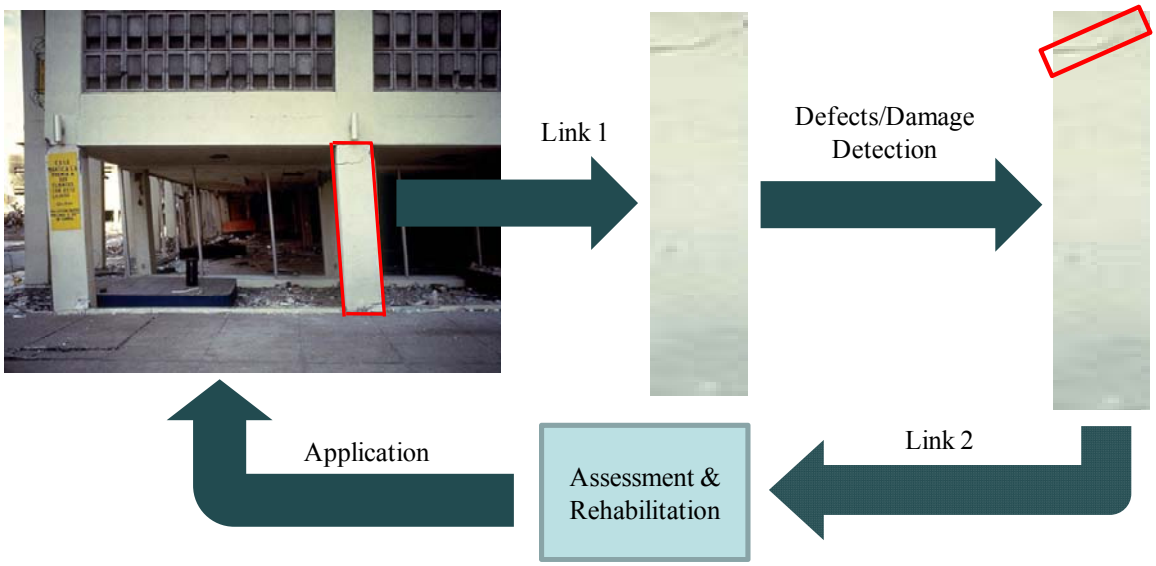


Figure 2: Two important links missing in existing research studies

1.2 Research Hypothesis

The hypothesis behind this research is: the missing two links (i.e. the connection between the defects/damage and the structural members and the relationship between the defects/damage and their impacts) can be automatically retrieved from images or videos by creating appropriate algorithms with the techniques in the areas of visual pattern recognition, digital filtering, and machine vision.

The emphasis in this research is placed on creating capable methods for 1) recognizing structural members in images and videos, and then spatially correlating the recognized structural members with the defects/damage detected by existing defects/damage detection methods (Link 1), 2) evaluating the impact of the detected defects/damage on structural members for automated civil infrastructure inspection, assessment, and rehabilitation (Link 2). The developed methods need to be validated in both controlled yet realistic settings and real built environments, and their corresponding performance needs to be measured with appropriate performance metrics.

1.3 Research Objectives and Scope

There are different types of structures in the world (e.g. concrete reinforced frames, wooden frames, etc.). One certain type of structures has categories of structural members (e.g. columns, beams, shear walls, etc.). Also, various defects/damage (e.g. cracks, spalling, air pockets, etc.) can be inflicted even on one structural member. Therefore, it is impossible to recognize all structural members in images/videos, and detect, spatially correlate, and evaluate all the defects/damage inflicted on the recognized structural members in one research study. Instead, the target of this research is old existing concrete structures. The research effort is focused on recognizing concrete columns in structures, since concrete columns are critical vertical load-bearing members. The defects and damage that are investigated in this research study include cracks, air pockets, and discoloration. Cracking is always an important structural damage indicator, while air pockets and discoloration are two primary influences affecting the aesthetic quality of a concrete surface.

The main objective of this research is to test whether the information about structural members (i.e. concrete columns in this research study) and defects/damage (i.e. cracks, air pockets and discoloration) can be retrieved from images/videos, and whether this information can be used to enhance the current manual visual inspection procedures for rapid assessment and rehabilitation of civil infrastructure. To support this main objective, specifically, the research effort is divided into the following three sub-objectives.

1. Create a novel method to recognize concrete columns in images and videos. The recognition is performed based on the columns' shape and material information. The recognition results are then compared with the concrete columns identified manually to indicate the effectiveness of the method.
2. Create a novel method to retrieve crack properties with a crack map produced by existing crack detection algorithms. The properties include crack width, length, and orientation. They are related to the dimension and orientation of the structural member to produce relative measurements. The results from the method are compared with those from manual surveys to find the measurement error on crack width, length, and orientation separately.
3. Create a novel method for automated assessment of the impacts of air pockets and discoloration on the visual quality of concrete surfaces in images/videos. The method includes locating air pockets and discoloration based on their visual characteristics, measuring the properties, such as size and area, of air pockets and discoloration, calculating the visual impact ratios based on the measured properties, and assessing the visual quality of concrete surfaces with visual impact ratios. The

assessment results from the method are compared with those from manual surveys to find assessment error.

4. Discuss the research findings, and describe the recommendations for the future research studies in rapid infrastructure assessment and rehabilitation.

This research study is limited to information retrieval from visual data (i.e. images and/or videos). Traditional documents, such as design drawings, are not considered, as information retrieval from the documents may follow another research topic. Similarly, three dimensional (3D) Computer-Aided Design (CAD) models and building information models (BIM) are assumed not available for existing old civil infrastructure, which was designed and built thirty or forty years ago. Both CAD models and building information models are more for newly-developed structures than existing ones. Previous research studies indicated that over two thirds of manual effort was still necessary to model even simple infrastructure (Sternberg et al. 2004; Jaselskis et al. 2005). The result is that these models are not produced in the vast majority of retrofit projects (Brilakis et al. 2011).

This research does not intend to enhance visibility in manual visual inspection. Instead, it aims to automating the current procedure of manual visual inspection and providing an evaluator with rapid and quantitative results at low inspection cost. Currently, the practice of manual visual inspection is performed only in structures where their load bearing members are at least partially exposed and visible. This condition is also held for the application of this research in practice.

This research involves the work of recognizing concrete columns, measuring crack properties, and assessing two common concrete surface defects: air pockets and discoloration. All the research work is performed in two dimensions (2D) instead of 3D.

Although there are many ways to address the problem of retrieving 3D information of a real world structure from multiple images and videos, it is often a time-consuming task, and requires different levels of user-assisted interactions (Larsen, 2010). Therefore, it does not satisfy the needs of automated rapid infrastructure assessment and rehabilitation.

1.4 Methodology and Results

In order to fulfill the aforementioned research objectives, the research work in this study includes 1) classifying concrete regions from images/videos, 2) recognizing column surfaces from concrete regions, 3) detecting cracks, air pockets, and discoloration in concrete regions, 4) measuring the properties of cracks, air pockets, and discoloration, 5) assessing the visual quality of concrete surfaces based on the properties, etc. All the research work has been organized into four levels, as illustrated in Figure 3.

In the first level, edges and concrete regions are retrieved from images and videos. Edges preserve the important structure properties of objects in images and videos, such as discontinuities in object surface orientation and changes in object surface material. They can be detected by checking the image areas with sharp image intensity changes. Concrete regions are classified with machine learning techniques. The visual features of each region are calculated and input into a pre-trained concrete material classifier. The output value of the classifier determines whether the region is composed of concrete.

The retrieved edges and concrete regions are then used to recognize concrete columns, and detect cracks, air pockets, and discoloration areas. A concrete column is recognized based on the condition of two long near-vertical lines at is sides and a concrete region in the middle. Cracks are detected according to their linear shapes, while air pockets are

detected by checking their circular shapes. Discoloration areas are located due to their special color characteristics.

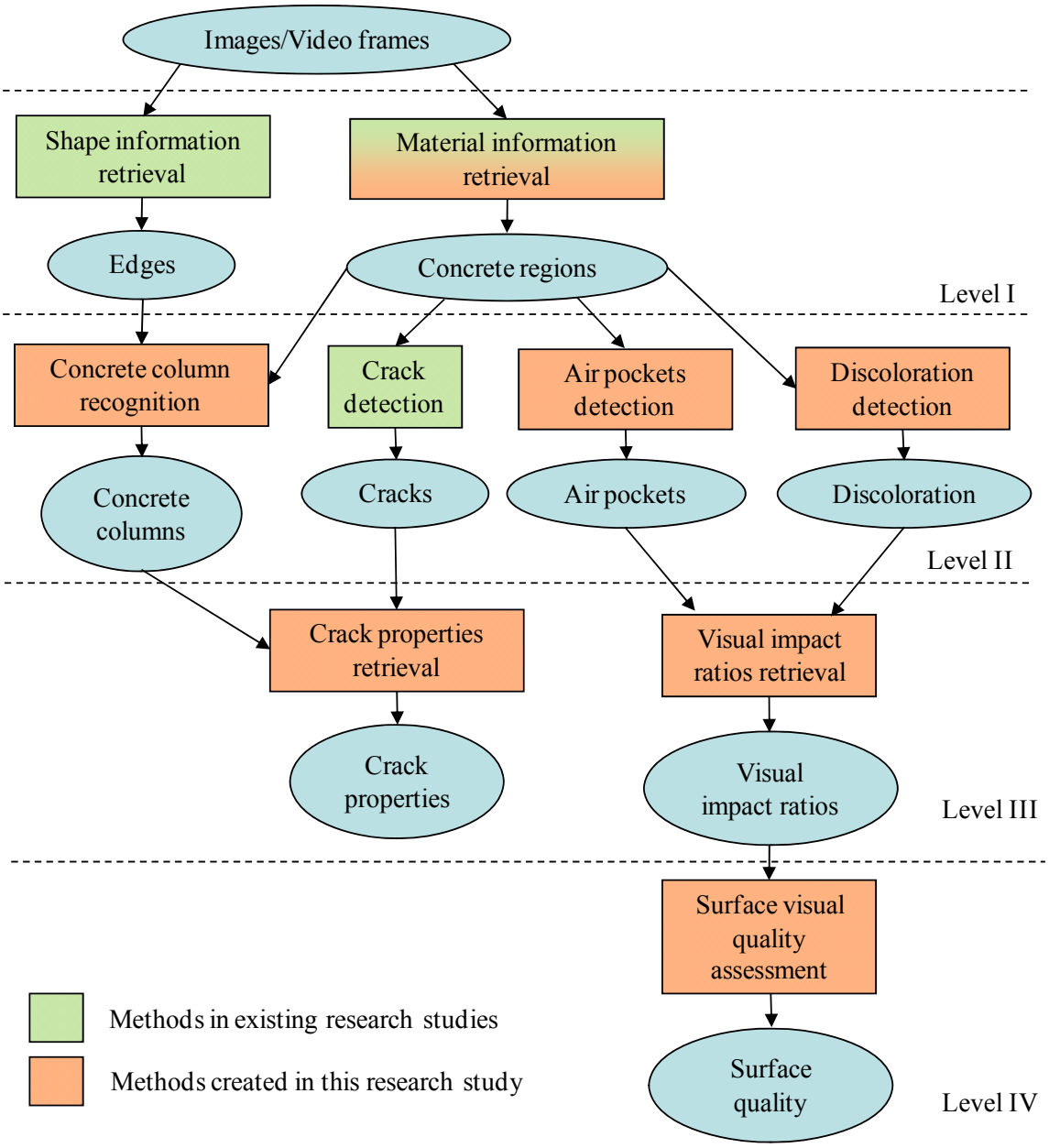


Figure 3: Methods created in this research study

The properties of cracks, air pockets, and discoloration are further calculated in the third level. The properties that are considered include the maximum width, length, and orientation of cracks, the size, area, and number of air pockets, and the area of

discoloration areas. All the properties are used for infrastructure assessment in the fourth level. For example, in this research study, the properties of air pockets and discoloration are converted into two visual impact ratios to assess the visual quality of concrete surfaces in terms of air pockets and discoloration. The properties of cracks are related to the dimension and orientation of concrete columns for assessing the damage states of the columns in terms of cracks.

The results from this research are measured with appropriate performance metrics. The accuracy of concrete region identification is defined as the percentage of the number of regions correctly identified within the total number of regions tested. Precision and recall are used to measure the performance of concrete column recognition and defects/damage detection. Precision is calculated as the percentage of the number of elements correctly recognized (or detected) within the total number of elements correctly and incorrectly recognized (or detected). Recall is calculated as the percentage of the number of elements correctly recognized (or detected) within the total number of elements correctly recognized (or detected) and not recognized (or detected) at all. The retrieved properties are compared with the properties from manual assessment to find the differences as property measurement error.

The test results validated the effectiveness of the methods created in this research study. The identification of concrete regions has high accuracy. The average precision and recall for concrete column recognition are 89% and 79%, while the average precision and recall for air pockets detection are 91% and 86%. The error in measuring crack properties can reach 2.21% in length measurement, 0.35% in maximum width measurement, and 3.29° in orientation measurement. The automated assessment of visual surface quality in terms

of air pockets and discoloration is almost matched with the manual evaluation from experienced inspectors. All the results indicate the effectiveness of using the techniques in image processing, digital filtering, and machine to automate and enhance the current manual visual inspection practices.

1.5 Contributions

The main goal of this research is to help inspectors and structural specialists make a well-informed decision for automated rapid infrastructure assessment and rehabilitation. The contributions of this research in routine inspection are listed as follows.

1. Enhance the manual visual inspection procedures by providing quantitative inspection results without the reliance on the experience and knowledge of one inspector and/or engineer.
2. Alleviate the challenge of not enough experienced inspectors in the architecture, engineering, and construction (AEC) industry, which is facing the pressing shortage of experienced and highly trained inspection and assessment personnel.
3. Save the amount of time and cost spent in the manual visual inspection procedures. The saved time and money can be reallocated by city or state agencies to initiate more inspection and maintenance projects to speed up the process of modernizing a large number of existing old structures with current limited budgets.

In an emergency scenario, the results and findings from this research can be used to:

1. Reduce the need of mobilizing structural inspectors/engineers to assess the safety of buildings after disasters, such as earthquakes. In the event of a moderate or large earthquake, it takes weeks or even months to recruit, train, and dispatch thousands of inspectors/engineers to inspect and evaluate damage structures. The

affected population is homeless and/or jobless during that time. The research can shorten the total inspection time and therefore alleviates the economical and societal impact of an earthquake.

2. Provide a crude yet rapid safety evaluation of the safety of entry into damaged structures for emergency responders. Emergency responders need to enter damaged buildings to save trapped victims within hours after an earthquake, but currently they have to wait for structural specialists to come and determine whether the damaged buildings are structurally stable to enter. The adverse effect is that the survival rate for trapped victims is significantly reduced. This research can automatically collect cracking information on concrete columns. The information can be transmitted outside through a local wireless network and the Internet to structural specialists at remote sites. The specialists watch the videos with the provided crack properties. When they perceive the potential building collapse, the responders can be informed to evacuate before the collapse happens.

The research work in concrete column recognition, and defects/damage detection can also be used to automate construction applications that include, but are not limited to:

1. As-built modeling. The current difficulty in as-built modeling lies in the retrieval of as-built information related to civil infrastructure, and then transforms it into an information-rich, object-oriented model. The process is human dependent to a great extent, and therefore, time-consuming and costly. This research provides examples of how to create visual pattern recognition (VPR) models that can automate the recognition of infrastructure-related elements, such as concrete surfaces, concrete columns, and the defects/damage inflicted on structures, based

on their visual features. The automatic civil infrastructure-related as-built information retrieval can facilitate the as-built modeling process.

2. Project progress monitoring at structural member level. This research can automatically recognize concrete columns in an as-built image. When the recognized columns are referred to the project as-planed models, it is convenient to figure out whether the project is ahead or behind the schedule.
3. Productivity measurement for construction operations. This research can automatically count how many columns are newly built from the videos captured in a certain period of time. This information can be used to measure the productivity of pouring concrete columns at a construction site.

1.6 Dissertation Organization

The motivation, hypothesis, objectives, methodology and results, and contributions behind this research have been introduced. The remaining chapters in the dissertation are organized as follows.

Chapter 2 is a background literature review chapter. It outlines the current practices of manual infrastructure assessment and rehabilitation in both routine and post-earthquake scenarios. This is then followed by an overview of the fundamental knowledge and previous research studies in structural member recognition, and defects/damage detection from images/videos, both of which this research plans to build on and augment. The chapter ends with an extensive summary on discussing the issues and limitations of manual visual assessment and previous research studies in rapid infrastructure assessment and rehabilitation.

Chapter 3 presents the first part of this research in designing a novel, automated method that can recognize concrete columns from images/videos. Each step of the

method is explained in detail. The performance metrics and factors that are used to measure the method are explained. Following that, the implementation and experimental test results of the method are presented. The chapter ends with an overview on the designed method, implementation, and experiments that are needed to prove the hypothesis of this dissertation research in critical structural member recognition.

Chapter 4 describes the second part of this dissertation research in defects/damage detection, properties retrieval, and evaluation from images/videos. The methods of retrieving crack properties with a provided crack map, detecting air pockets and discoloration on concrete surfaces, and evaluating the impacts of air pockets and discoloration on the visual quality of the surfaces are explained separately. The performance metrics and factors that are used to measure these methods are introduced. Following that, the design of the experiments that validate these methods is presented. The chapter ends with an overview on the method descriptions, implementation, and experiments that are needed to prove the hypothesis of this dissertation research in critical defects/damage detection, properties retrieval, and evaluation.

Potential application fields of this dissertation research in construction, structural engineering, and transportation are introduced in Chapter 5. In Chapter 6, the findings and contributions of this research are described. The chapter ends with a discussion on future research directions. The thesis concludes with the references, the appendices, and the curriculum vita of the researcher.

CHAPTER 2
LITERATURE REVIEW

This chapter first outlines the current practices of manual visual inspection in both routine and emergency scenarios. Recent research efforts in rapid infrastructure inspection and assessment are then presented. These two are followed by an overview of the fundamental knowledge in the classification of construction materials, the recognition of structural members, and the detection of concrete surface defects/damage, all of which this research plans to build on and augment.

2.1 Manual Visual Inspection

Although the procedures of inspecting different types of civil infrastructure in routine or emergency scenario are various, the manual inspection of concrete products typically involves: 1) walking through the inspection area; 2) gathering information on the design, construction, and ambient conditions of the structure; 3) planning the complete investigation; and 4) laying out a control grid for recording observations (ACI 228.2R-98, 2005). After inspection, the condition of the finished product is evaluated qualitatively (good, satisfactory or poor) (ACI 201.1R-92, 2005).

In order to help the readers acquire a deeper understanding of the current practice in manual visual inspection, the specific inspection procedures for highway bridges and post-earthquake buildings are presented. This is then followed by the discussion of the issues and limitations associated with the manual visual inspection practice.

2.1.1 Current Practices in Routine Highway Bridge Inspection

Most highway bridges are required to be inspected at regular intervals (usually not exceeding two years with few exceptions) to determine their physical and functional conditions and ensure that they satisfy present service requirements (AASHTO, 2001). The work is currently carried out manually by certified inspectors following the established standards and manuals (i.e. the National Bridge Inspection Standards and AASHTO Manual on Bridge Evaluation). Before going onto a bridge, the inspectors need to prepare sketches and note templates for references throughout the inspection (Sunkpho, 2001). During the inspection, they record the actual bridge conditions by observing existing defects that lay on primary bridge components, such as decks, exterior beams and piers. The defects to be inspected include different types of cracks (e.g. flexural, shear, vertical and bond cracks), loss of cover and spalling, and corrosion and efflorescence. This information is used to rate bridge conditions.

So far, there are two rating systems that are adopted in practice. The FHWA Recording and Coding Guide (1995) defines one system which uses the National Bridge Inventory (NBI) zero to nine scale for rating a bridge deck, superstructure, and substructure (Figure 4a). The second system, the PONTIS rating system, uses a set of three to five condition states to describe the condition of approximately 160 bridge elements, such as columns, girders, slabs and trusses (Thompson and Shepard, 2000) (Figure 4b). Both rating systems are built on qualitative definitions. For example, in the NBI rating system, a bridge is rated at “Fair Condition” (Condition State 5), when all of its primary components are sound but may have minor cracking, spalling, or scour (FHWA, 1995). In the POINTS rating system, reinforced concrete elements are identified

in “Fair” (Condition State 2), when minor cracks and spalls may be present but there is no exposed reinforcing or surface evidence of rebar corrosion (MDOT, 2007).

Bridge	Component	Rating
B 531	Deck	6
	Superstructure	6
	Substructure	6

(a) NBI rating system (Phares et al. 2004)

Bridge 9340 I 35W OVER RR, MISS R, 2ND ST & RD INSP. DATE: 06-15-2006								
Element #	Element Name	INSP. DATE	Quantity	Quality				
				1	2	3	4	5
205	Concrete Column	06-15-2006	52 EA	49	3	0	0	N/A
		06-10-2005	52 EA	49	3	0	0	N/A
210	Concrete Pier Wall	06-15-2006	168 LF	168	0	0	0	N/A
		06-10-2005	168 LF	168	0	0	0	N/A

(b) POINTIS rating system (MnDOT, 2007)

Figure 4: Examples of highway bridge rating systems

2.1.2 Current Practices in Post-Earthquake Building Assessment

After an earthquake occurs, it is necessary for people to enter damaged buildings due to a variety of reasons, including emergency search and rescue, building stabilization and repair, and salvage and retrieval of possessions (ATC-35, 1999). However, potential structural collapse in an aftershock often places them in danger and produces additional victims. In order to reduce this risk, the Federal Emergency Management Agency (FEMA) requires that emergency search and rescue teams can only enter the buildings that are

determined structurally stable (FEMA, 2009). The damaged buildings also need to be classified into different categories (SAFE, RESTRICTED USE, or UNSAFE) to inform owners, occupants, and the public about their conditions in terms of their suitability for future occupancy and general use (ATC-35, 1999).

Currently, the safety evaluation of buildings in the event of an earthquake is performed by structural specialists, such as certified inspectors and/or structural engineers. The involved structural specialists are responsible for identifying potential structural hazards and monitoring the structure for condition changes (FEMA, 2006). They follow existing procedures and/or guidelines, and make an assessment based on their experience and knowledge coupled with their visual observation of the damage inflicted on the load-bearing members of a structure. For example, in the case of evaluating the safety of post-earthquake structures, the ATC-20 (1989) and ATC-20-2 (1995) codes outline two procedural levels: rapid evaluation and detailed evaluation.

The rapid evaluation is typically based on an exterior inspection of a structure only, and its purpose is to quickly identify apparently "Unsafe" or "Safe" buildings after an earthquake. In the procedure of the rapid evaluation, a structure is determined as "Unsafe", when the following building conditions are observed: 1) collapse, partial collapse, or building off foundation; 2) building or story leaning; 3) racking damage to walls, other structural damage; 4) chimney, parapet, or other falling hazard; and 5) ground slope movement or cracking (ATC-20, 1989).

The detailed evaluation is a thorough visual inspection of a structure inside and outside. Its purpose is to classify the buildings that cannot be determined as "Safe" or "Unsafe" after a rapid evaluation. In the detailed evaluation, inspectors and/or engineers

do not only consider overall building hazard, but also the severity and extent of damage to the structural members, such as columns, walls, roofs, and precast connections. For example, a cast-in-place concrete building is determined as "Unsafe", when any of the following three conditions exists: 1) buckled or fractured columns; 2) exposure of vertical column reinforcement; or 3) large diagonal cracks extending though columns.

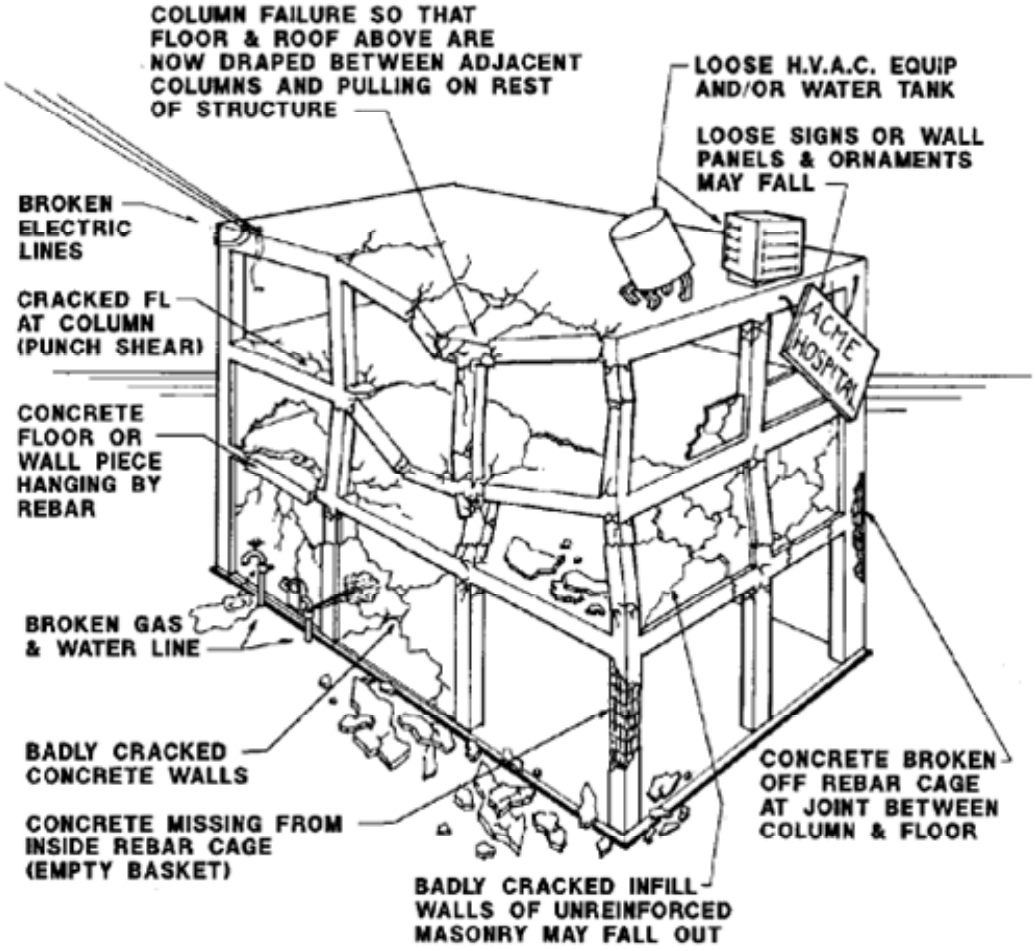


Figure 5: Collapse patterns and check points (FEMA, 2009)

In addition to ATC-20 and ATC-20-2 documents, Federal Emergency Management Agency (FEMA) also offers structural collapse technician courses and training seminars. The courses and seminars provide the basics of structural collapse patterns and common check points in a structure (Figure 5) (FEMA, 2009). This facilitates the training of the

personnel, who do not have enough civil or structural engineering background, to be aware of all visible and critical structure damage, when they encounter in a post-earthquake building.

When the rapid evaluation and the detailed evaluation cannot classify a structure as "Safe" or "Unsafe", engineering evaluation is necessary. In the engineering evaluation, inspectors and/or engineers investigate the safety of a structure from its construction drawings and new structural calculations in detail, which always takes at least one week to complete (NASA, 2009).

2.1.3 Issues and Limitations of Manual Visual Inspection

There are several issues or limitations in the practices of manual visual inspection that have been identified from previous research studies. First, manual inspection is inefficient and time-consuming. Inspectors need to become familiar with the structure. Reviewing previous reports, preparing inspection plans, and collecting and analyzing field data may require a large amount of inspector hours, which has proven costly over time (Bartel 2001; Jauregui and White, 2003). In the event of a disaster, such as an earthquake, the inspection process may take weeks or even months to complete, when the lack of available inspectors in the affected area is combined with the large volume of damaged buildings. According to the report about the October 15, 2006 Hawaii Earthquake, over several hundred buildings were requested to assess damage each day from October 15 to the end of October in the County of Hawaii, but only approximately 100 of them could be evaluated (Chock, 2007). Similar findings were also reported by Johnson for the December 22, 2003 San Simeon Earthquake, where the phases of assessment work continued for a number of weeks (Johnson, 2004). The long elapsed

time spent in evaluating post-earthquake buildings makes humans exposed to precarious working and living conditions, which directly translates into significant economic losses (Kamat and El-Tawil, 2007).

Second, the results from manual visual inspections are subjective and highly variable. An on-site survey was conducted by the FHWA Non-Destructive Evaluation Center (NDEVC). In the survey, 49 bridge inspectors from 25 states were invited to inspect 7 bridges. The results indicated that 32 percent of condition ratings varied more than one rating point of the average in the condition rating system defined in the FHWA Recording and Coding Guide (Moore et al. 2001). Take one inspected bridge, B521, for an example. The minimum ratings for its deck, superstructure, and substructure are 3, 4, and 3, while the corresponding maximum ratings are 7, 8, and 7 (Figure 6). This means that some inspectors considered that the primary bridge components experienced POOR (rating 4) or even SERIOUS (rating 3) conditions, while some inspectors claimed that the bridge components still sustained GOOD (rating 7) or VERY GOOD (rating 8) conditions, according to the condition rating system defined in the FHWA Recording and Coding Guide (1955). Similarly, Paterson et al. (1997) also revealed that current manual inspection techniques are subjective and prone to errors in building inspection.

Bridge B521			
	Deck	Superstructure	Substructure
Average	5.8	5.9	6.1
Minimum	3	4	3
Maximum	7	8	7
# of Inspectors	49	49	49

Figure 6: Condition ratings for Bridge B521 (data from Phares et al. (2004))

Third, the requirement of experienced inspectors in manual inspection poses a challenge for the architecture, engineering and construction industry. The industry is facing the pressing shortage of experienced and highly trained inspection personnel. On the one hand, the industry has difficulty in attracting and retaining young inspectors (COMT, 2008). On the other hand, as early as in 1995, Prine pointed out that many experienced inspection personnel were given early retirement, but they were not replaced by entry level personnel with minimal experience and training (Prine, 1995). This shortage of skilled inspectors pushed up the salaries of experienced inspectors. Owners or general contractors have to pay sizable inspection fees due to the significant compensation of experienced inspectors. For example, according to the data provided by the city of Lincoln (2007) in Canada, an experienced inspector can earn approximately 43% more than a less-experienced inspector.

Moreover, although inspectors and/or engineers are the appropriate candidates to evaluate built environments in urban areas (Aldunate et al. 2006), there are several issues when they are involved in the emergent efforts of relieving disaster impacts with other organizations. Kostoulas et al. (2006) identified the collaboration-related problems between the inspectors/engineers and other emergency responders, such as fire fighters, due to the lack of coordination, information sharing, trust and communication. Also, there are not enough qualified inspectors/engineers that can be allocated to community emergency response teams. Those that can participate must be licensed professional engineers with a minimum of five years of experience. In addition, they are required to take structural collapse technician courses and US&R structural specialist training. In large-scale disasters where several thousand buildings may be affected, evaluating the

safety of all buildings manually by a limited number of structural specialists requires a significant amount of time (Mehta and Pena-Mora, 2009).

In order to address the aforementioned issues and limitations, it is necessary to automate the current manual evaluation practices. Recent research efforts are mainly focused on the automatic and fast retrieval of building and damaging information.

2.2 Recent Research Efforts for Automated Rapid Infrastructure Assessment

Existing research efforts for automated rapid infrastructure assessment can be generally divided into two categories. The methods in the first category mainly rely on the sensing data (e.g. stress and strain data of structural members) from pre-installed structural health monitoring (SHM) sensors to determine structure damage states. Kottapalli et al. (2003) showed that sensor networks installed in new buildings can provide useful information for structural damage evaluation. Naeim et al. (2005) also tested the feasibility of using SHM sensing data to evaluate damaged buildings. According to their test results on seventy buildings, they indicated that that such data can be used for this purpose through wavelet analysis, fuzzy logic analysis, or probabilistic approaches with fragility functions (Naeim et al. 2005). Tsai et al. (2007) designed a building black box system that can automatically collect SHM sensing data, along with building design and construction information, such as floor plans, materials, deployment of electrical and plumbing systems. This information is then provided to structural specialists, when they are deployed to support emergency responders in urban disaster response and relief efforts (Tsai, 2007). Although the SHM sensing data can reflect the current state of a building, sensor networks are now only pre-installed into a small

percentage of existing structures in earthquake prone areas and most critical civil infrastructure in a nationwide area.

Instead of the reliance of a pre-installed SHM sensor network, Kamat and El-Tawil (2007) proposed a rapid buildign safety evaluation method based on augmented reality. The method proposed by them operates by superimposing a Computer Aided Design (CAD) image over the user's view of the real world, so that the permanent deformed shape of a building component can be detected and measured as sotry drift ratios (Kamat and El-Tawil, 2007). The evaluation is then performed based on the story ratios of a whole structure. The method requires the 3D CAD model of a building and the known locations of fiducial marks for CAD image superimposition. These two prerequisites limit the applicability of the method in practice. The vast majority of existing structure in use were built thirty or even forty years ago. They need the assessment more than their modern counterparts, but they were designed with 2D drawings instead of 3D CAD models. A lot of manual efforts are necessary to create as-built 3D CAD models for these existing old structures. Sternberg et al. (2004) and Jaselskis et al. (2005) noted that over two thirds of manual efforts were spent on manually converting the collected spatial data to a 3D model, even when modeling simple civil infrastructure. Similar findings were also reported by professional modelers, such as Reality Measurements (2009), and in the researcher's work of comparing different spatial data collection techniques for civil infrastructure modeling (Zhu and Brilakis, 2009).

Although the methods in both categories can complement current manual practices by providing a non-subjective and quantitative assessment, they can only be used when the structures are instrumented with structural health monitoring sensors or have 3D CAD

models readily available. This precludes their applicability either in the routine or emergent practice. Today, manual visul inspection is still widely adopted.

In order to enhance the current mnaul visual inspection process, one way that has been proved effectively so far is to use machine vision techniques to enhance and/or replicate the current manual visual inspection. Suppose that the damaning information on structural elmenets of a structure can be detected from images/videos. Most limitations associated with the manual visual inspection (e.g. subjective nature of inspection results, time-consuming evaluation process, and not enough qualified inspectors/engineers) can be overcome. In doing so, two critical components in visual inspection need to be autoamted. The first one is the recognition of structural members from imges and videos, and the second one is the detection of defects/damage that are inflicted on the surfaces of structural members. The following two sections describe existing methods that have been developed in these two areas.

2.3 Structural Member Recognition

The retrieval of structural member information in images and videos is always the first step to facilitate many construction applications, including monitoring project progress, measuring construction productivity, improving construction safety, and evaluating building safety. For example, in assessing the progress level of a project, constructed facilities need to be first distinguished and extracted. The identified as-built facilities were then compared with the project's as-planned 3D CAD model (Choi et al. 2008; Wu and Kim) or 4D augmented reality model (Golparvar-Fard et al. 2010) to indicate whether the project is behind and ahead of the schedule. In productivity measurement, Gong and Caldas (2009) investigated the behavior of a recognized bucket in a video to

analyze the cyclic construction productivity of concrete pouring. In pro-active construction safety protection, Teizer et al. (2007) introduced an occupancy grid modeling method to model, detect, and track both moving and static objects (e.g. a worker vs. an electrical pole), so that the potential collapse between them can be identified in advance.

Currently, the information of structural members is retrieved manually from images and videos. Site engineers have to label each image or video based on their interpretation of the contents of the image or video first. Then, the structural member information can be retrieved through label searching and matching later. The label is composed of certain "keywords", such as concrete columns, beams and slabs. In order to facilitate the process of labeling an image or video, Kosovac et al. (2000) proposed the use of standard thesauri to assist site engineers in selecting standard "keywords" from the thesauri to compose the label. Abudayyeh (1997) also developed a prototype relational database using Microsoft Access to facilitate labels searching and matching electronically. Although the retrieval of structural member information becomes easy when the labels of the images and videos are created, manually labeling each image or video is time-consuming and tedious. Assuming only five to ten images or videos are collected daily in a project, the task of labeling them requires a lot of patience (Brialkis et al. 2006). For this reason, automated structural member recognition has been recently investigated.

Given the template of a structural member, structural member recognition in general is regarded as the problem of locating the element that "looks" similar to the template (Ge et al. 2008). The template in structural element detection from images and videos can be a set of color/texture regions, boundaries or other image features that are invariant to

scale/affine transforms. Depending on the different type of the template adopted, existing automated structural element detection methods from images and videos are classified as color/texture-based, shape-based and scale/affine-invariant feature -based.

2.3.1 Color/Texture-Based Methods

Color and texture are always used as an effective indicator for structural member recognition, since they provide a good and detailed surface description of structural members in an image or video especially when their color and texture are unique. One common color model widely used is the RGB color model, which has three primary components: Red, Green, and Blue. These three components are added together independently to reproduce a broad array of color values in various ways. As for texture, it can be retrieved from filter banks, such as the Leung-Malik (LM) (Leung and Malik, 2001), the Schmid (S) (Schmid, 2001), and the Root Filter Set (RFS) (Varma and Zisserman, 2005).

A primary step in the recognition of structural members using color and texture is to form a decision boundary in the feature space of color and texture to determine whether the input color and texture belong to the structural members. This task is typically referred as a binary (two-class) classification, since the boundary is always made from both sides by the examples of color and texture values in two groups (e.g. concrete vs. wood, steel, brick, etc.) (Tax, 2001). However, creating the boundary specific for one type of construction material may be not a simple binary classification problem. This is because, on the one hand, the samples in the target class (e.g. positive concrete samples) are always available. On the other hand, the samples in the outlier class (e.g. negative concrete samples including wood, brick, earth, sky, grass, sand, etc.) are abundant and

diverse. As a result, it may be impossible to provide "typical" negative samples to characterize their distribution (Kwak and Oh, 2009).

So far, much effort has been expended to solve both one-class and two-class classification tasks. The common techniques include Gaussian mixture modes (GMMs), k-nearest neighbor (k-NN), boosting, support vector machine (SVM), artificial neural networks (ANN), etc. Tan and Gilbert (2003) found that statistical classification methods (e.g. SVM, neural networks) tended to perform better than rule-based ones (e.g. decision tree) over multi-dimensions and continuous data attributes. Tax (2001) compared different classification methods concerning their robustness to outliers, the ease of classifier configuration, and computation requirements. It was observed that SVDD (one type of SVMs) is more robust to the outliers in the training data than GMMs and K-nearest neighbor (Tax, 2001). Ratsch et al. (2002) showed that boosting is almost equivalent to SVM, since a boosting algorithm can be constructed from a SVM algorithm and vice versa. Also, Kotsiantis (2007) found that both SVMs and ANNs performed better than other machine learning techniques in the supervised learning; however, neither of them is perfect. On the one hand, SVMs can guarantee a global optimum solution, while ANNs can only guarantee local optima; on the other hand, ANNs are fast in classification and robust to noise (Li, 2010).

Specifically for the recognition of structural member using color and texture cues in the field of civil engineering, Neto et al. (2002) first observed that most construction materials, such as steel and concrete, do not have a constant color value but a range of nuance of the value due to the variance of light conditions or the nature of the materials. Based on this observation, their method started from the identification of the boundaries

of structural members by tracing each image pixel that satisfies two conditions: 1) the color value of the pixel fits in a color value range; and 2) the color value of at least one of its neighbor does not fit in the color value range. When the boundary of a structural member has been identified, all internal pixels in the boundary are checked. Those whose color values do not fit in the color value range are removed. Both the boundary and the remaining internal pixels represent one structural element. In their work, the color value range of structural elements has to be manually pre-defined. Also, their method can only detect the presence of structural elements in images, but fail to classify the type of the elements as columns, beams or walls.

Instead of using color information only, Brilakis et al. (2006) developed a method of retrieving material information in an image using the concept of material "signature". In their method, an image is first cropped into regions using bottom-up clustering methods. The signature of each image region is then calculated and represented by a vector of the mean and the standard deviation of region's color and texture values (i.e. intensity, normalized red, green and blue, six response values to bank filters). The signature is compared with the signatures of material samples stored in a material knowledge database by measuring their Euclidean distances. If the distance of the region's signature to the signature of one material sample is smaller than a pre-defined threshold, the region is assumed to contain that material. For example, if the signature of one image region is matched with that of a concrete sample in the database, it assumes that the material contained in the region is concrete.

Extending from their material recognition work, Brilakis and Soibelman (2008) further developed a method of recognizing linear structural elements. When all material

regions in an image are identified, they calculated the maximum cluster dimension (MCD) and the maximum dimension along the perpendicular axis of MCD (PMCD) of each material region. The region is classified as a column (or beam) based on three assumptions: 1) the region is linear, if its MCD is significantly larger than PMCD; 2) if the region is linear, then the tangent of the MCD edge points represents its direction on the image plane; and 3) the linear region is a column (or beam), if the computed direction is within 45° from the vertical (or horizontal) image axis (Brilakis and Soiblman, 2008).

Although the three assumptions are valid, the sole dependence on material information makes the color/texture-based method not always work in detecting structure elements in most frame structures. For example, when one structural element (e.g. a concrete column) is connected to another structural element with the same material (e.g. a concrete beam), which is common for reinforced concrete frame structures, the method regards them as one single element instead of two separate elements (Figure 7).

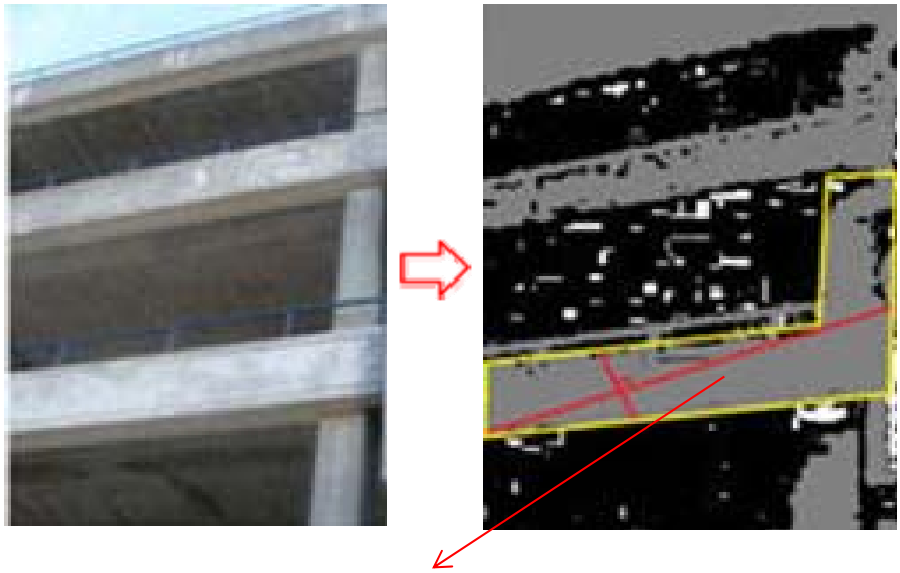


Figure 7: A limitation of detecting structural members using material information

2.3.2 Shape-Based Methods

In addition to color and texture, edge information is another indicator for the detection of structural elements. Edges in images are always at the areas with sharp image intensity changes. They preserve the important structure properties of the elements in images, such as the discontinuities in element surface orientations and the changes in element surface materials (Green, 2002). The properties are useful and appropriate in detecting thin and stick-like structural elements, since their color and texture are easily corrupted by image background (David, 2005).

Several shape-based methods have been developed to detect rectangular elements in images/video frames. These methods, with few exceptions, start with edge detection using low level image processing operators, such as Canny or Sobel (Green, 2002). Other active contour-based edge/boundary detection methods are not considered here, since these methods usually require a manual contour initialization step and a full mathematical and analytical description of contour lines (Lankton and Tannenbaum, 2008). Canny operator is preferred due to its optimal design and satisfactory detection results that can be retrieved (Guo et al. 2009). The results from Canny or Sobel operators are only isolated edge points without any line information. It is unknown which group of points should be linked to form a line, until line detection algorithms are used.

The Hough Transform is one of such line detection algorithms. It is powerful in detecting lines, but its brute force voting scheme for each detected edge point once imposed a heavy computation cost. In order to improve its efficiency, several research efforts have been made. For example, Bandera et al. (2006) introduced the mean shift based clustering stage into the Hough Transform. The clustering stage avoids the global

maxima detection in the Hough Transform, which makes it run faster. Fernades and Oliveria (2008) recently replaced the brute force voting scheme of the Hough Transform with the elliptical Gaussian kernel based one. Now the real-time performance of the Hough Transform can be achieved even for a high resolution image with these efforts.

Aside from the Hough Transform, lines can also be extracted based on edge points' covariance matrices (Guru et al. 2004) or principle component analysis of their distributions (Lee et al. 2006a). However, the former needs to manually estimate the proper size of a mask which is moved from one pixel to another to generate covariance matrices. The latter requires a lot of time in labeling edge points. The edge labeling errors, such as the same label of an edge point for two straight lines meeting together with a small angle, leads to the failure of the method. Both methods cannot work when excessive image noise pixels exist.

The results of line detection can be used to detect 2D rectangular elements (e.g. vehicle license plates) in images. Jung and Schramm (2004) found that a rectangle in Cartesian space corresponded to four peaks in Hough space from its two pairs of parallel lines. Based on this finding, their method tried to extract a 2D rectangular element in an image by searching its corresponding four peaks in the Hough space. Preliminary results indicated that the method can detect 2D rectangles with varying sizes and orientations when appropriate parameters are set. However, it is of no use in detecting rectangular elements in the real world, which are usually projected in images as quadrilaterals instead of rectangles.

In order to detect rectangular elements in the real world, the vanishing points of the retrieved lines need to be estimated first. The estimation is performed mainly through line

clustering (Rother, 2000), Gaussian sphere (Cantoni et al. 2001) or linear least square minimization (Kosecka and Zhang, 2002). When the vanishing points are found, quadrilaterals are detected through exhaustive line pair matching (Shaw and Barnes, 2006) or graph-based search (Micusik et al. 2008). This way, multiple quadrilaterals in an image or video frame can be detected, even if they are partially occluded. However, the assumption of the presence of vanishing points in each image or video frame is not always true (Almansa et al. 2003). In addition, the process of detecting quadrilaterals in these methods takes tens of seconds, which makes them not appropriate for near real-time applications. Also, only quadrilaterals whose sides are aligned with the dominant scene directions in an image or video frame can be detected.

Although the results of the methods of detecting rectangular elements are promising, the sole reliance on edge information makes the methods not always robust. For example, in Figure 8, it can be seen that steel rails at the top of the figure were recognized as concrete columns, because they had similar shapes of the columns. Also, it was found that the methods solely based on edge statistical analysis (i.e. high edge magnitude and variance) could fail in complex scenes (Nikolaos et al. 2006). The failure might be produced, when the boundaries of the elements are not clear due to damages or dirt (Abolghasemi and Ahmadyfard, 2008).

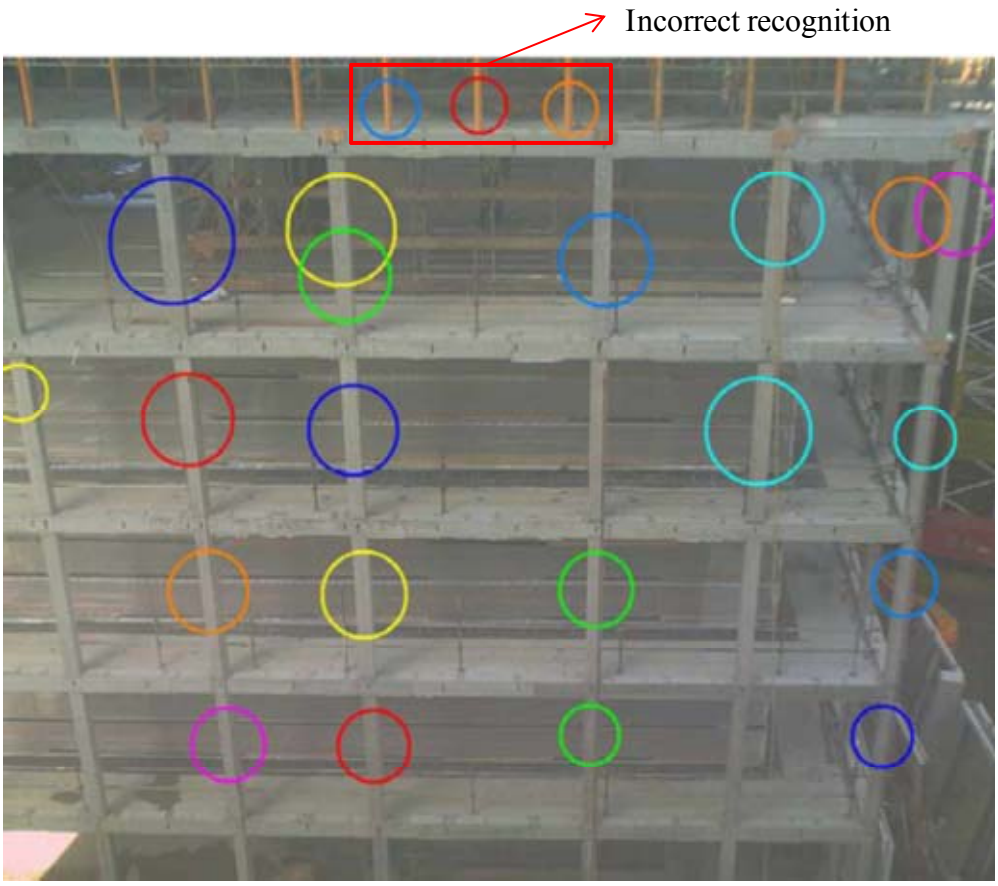


Figure 8: A limitation of detecting structural members using shape information (data from Lukins and Trucco (2007))

2.3.3 Scale/Affine-Invariant Feature-Based Methods

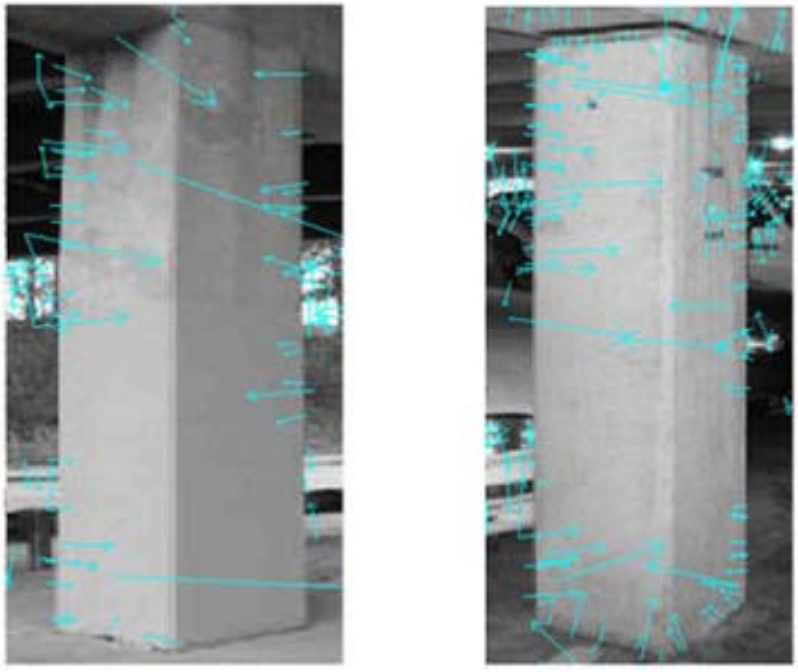
Color/texture-based methods utilize color and texture values to locate an element, while shape-based methods detect an element based on its shape information. Scale/affine-invariant feature-based methods identify an element based on a set of image features that describe the characteristics of the element in images/videos, and are also invariant to scale and/or affine transforms. The general process of the methods includes two important stages: 1) feature extraction and 2) matching (Scordino, 2006).

The purpose of feature extraction is to provide a useful element description that can characterize the element in an image or video frame (Lin et al. 2004). The description is invariant to scale, rotation, and preferably changes in light conditions (Cornelis et al. 2008). The scale invariant feature transform (SIFT) is one of such algorithm developed towards this goal (Lowe, 2004). It uses a 3D histogram of gradient locations and orientations to represent a local image region. The quantization of gradient locations and orientations makes the SIFT robust to small geometric distortions (Mikolajczyk and Schmid, 2005). In addition to SIFT, other algorithms, such as RIFT (rotation-invariant generalization of SIFT) (Lazebnik et al. 2004), PCA-SIFT (principal component analysis of SIFT) (Ke and Sukthankar, 2004), and GLOH (Gradient location-orientation histogram) (Mikolajczyk and Schmid, 2005), can also be used for the purpose of element features extraction.

Feature matching is to establish the correspondence between the features extracted from the current image or video frame and the features extracted previously from the template of an element. Its purpose is to find whether the feature vectors extracted from the current image or video frame contain the features extracted from the element template. If yes, it means that the element exists in the current image or video frame. Feature matching can be defined as a nearest neighbor search problem (Muja and Lowe, 2009). The problem can be solved through the use of multiple randomized k-d trees (Silpa-Anan and Hartley, 2008), a spill tree (Liu et al. 2004) or a hierarch k-means tree (Nister and Stewenjus, 2006).

Scale/affine-invariant feature-based methods are powerful in detecting a specific element in multiple images or video frames, but they are not appropriate for the detection

of elements in one category. For example, although concrete columns in images or videos, are geometrically simple, they are characterized by large topographical variations, such as aspect ratios. Therefore, no simple scale/affine transformation can characterize them. The extracted "features" only reflect the individual characteristics of each column but not the salient features of the column category. Second, few distinctive local features can be retrieved from the surfaces of structural elements, since they are always made of uniform materials. The uniform materials make existing scale/affine-invariant feature-based methods difficult to find local features on the element surface. Third, the existence of dirt on the surfaces of structural elements results in the false local features produced for structural element detection. All these limitations have been clearly illustrated in Figure 9, where the arrows indicate the location, gradient and orientation information of the key-points detected by the SIFT algorithm.



(a) SIFT features for column A (b) SIFT features for column B

Figure 9: Different SIFT features for two concrete columns

2.4 Defects/Damage Detection

When structural members are recognized from images and vides, the next step is to detect the damage inflicted on the structural members in order to evaluate the conditions of the members. Although manual visual inspection is still a popular inspection method, automated visual inspection using image processing and machine vision techniques has been recently developed (Lee et al. 2006b). For example, Guo et al. (2009) proposed a method of recognizing abnormal regions at the surfaces of concrete pipes. In addition to the recognition of abnormal regions, the detection methods for specific types of defects anddamage have also been created based on image processing techniques, such as wavelet transforms, Fourier transforms, thresholding, edge detection, and/or region-based segmentation. Their effectiveness has been verified in detecting cracks, coating rusts, and air pockets, when inspecting structures like bridges, underground pipes and tunnels. The highly successful efforts validated the ability of image processing and machine vision techniques in detecting damage and defects for civil infrastructure related identification and assessment problems, even when well-light conditions are not available (e.g. underground pipes).

2.4.1 Crack Detection

Cracking is always an important structure damage indicator, and is therefore listed into the inspection check-list no matter in a routine or emergency scenario. Take highway bridge routine inspection for an example. According to the National Bridge Inventory (NBI) rating system adopted in the FHWA Recording and Coding Guide, a bridge is rated at “Fair Condition” (condition five), when all of its primary components, such as columns, girders, slabs, and trusses, are sound but may have minor cracking, spalling, or

scour (FHWA, 1995). In the post-earthquake inspection for emergency responders, one check point for concrete frame structures is whether cracking occurs on concrete columns at each floor line (above and below floor) in the educational materials for the structural collapse technician courses provided by the Federal Emergency Management Agency (FEMA, 2009). In the ATC-20 code (Procedures for Post-earthquake Safety Evaluation of Buildings), the extent and severity of damage to the load-bearing elements of a reinforced concrete building is quantified primarily by the width and orientation of the cracks that lie on these elements (ATC-20, 1989). A cast-in-place concrete building is immediately regarded as unsafe and prohibited to entry when there are large diagonal cracks extending though building columns (ATC-20, 1989).

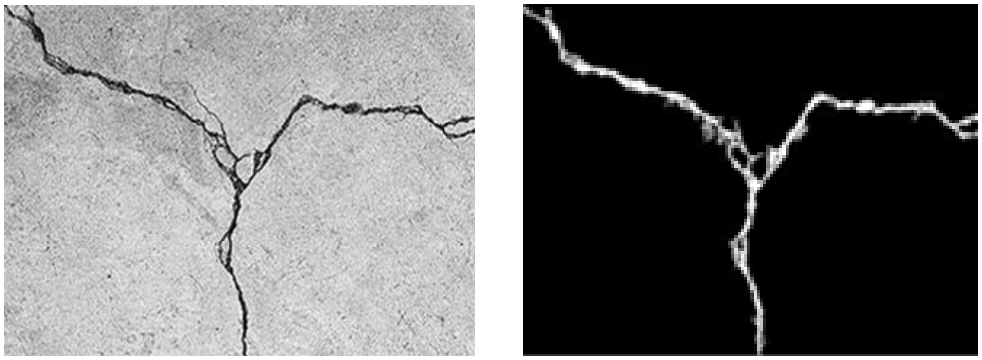
So far, many methods have been created to automatically detect the cracks inflicted on the images of structural element surfaces. They are generally classified into two categories. The methods in the first category only recognize whether or not the image of a structural element surface contains a crack (i.e. crack presence detection). For example, Abdel-Qader et al. (2006) proposed a principal component analysis (PCA) based method to identify the presence of the cracks in a bridge surface image. In their method, an image was first segmented into sixteen square blocks. Each block was filtered by linear feature detectors (horizontal, vertical and oblique) and then projected onto dominant eigenvectors which were pre-generated from a training data set containing crack and not-crack blocks. The projection result was further compared with the projection results of the training data to identify whether or not the blocks contain cracks (Figure 10). Liu et al. (2002) developed a crack classification system, where a support vector machine (SVM)

based classifier was used to classify each region in an image as one of three types: "crack," "non-crack" or "intermediate".

Test block	Pre-identified block	Distance	Results
11	9 (Non-crack block)	4.739	Non-crack
12	9 (Non-crack block)	4.3598	Non-crack
13	1 (Crack block)	8.7912	Crack

Figure 10: Crack presence determination (data from Abdel-Qader et al. (2006))

The methods in the second category can not only detect the presence of cracks in an image, but also locate crack points in the image and produce a crack map. The crack map is a binary image. The crack points in the image are typically marked white and non-crack points (surface background) are black (Figure 11).



(a) Original image (b) Crack map

Figure 11: A crack map example

A lot of image processing techniques can be used to produce crack maps, including wavelet transforms, thresholding, and edge detection, to identify crack points from the image background. For example, Cheng et al. (2003) detected cracks in an image by simply thresholding the concrete surface image. The threshold value was determined based on the image's mean and standard deviation values. Abdel-Qader et al. (2003)

compared the effectiveness of four edge detection techniques (the Canny edge detector, Sobel edge detector, Fourier transform and fast Haar transform) with respect to the detection of cracks on concrete bridges and found that the fast Haar transform was more reliable than the other three. All these methods belong to global-processing techniques without considering any crack connectivity information. As a result, the detection accuracy of these methods is easily affected by image noise.

In order to address this problem, Sinha and Fieguth (2006) introduced two crack detectors that consider relative statistical properties of adjacent image regions. These two detectors are applied in four directions (0°, 45°, 90°, and 135°) to identify crack pieces in buried concrete pipes, and then a linking and cleaning algorithm is used to connect crack pieces. Iyer and Sinha (2006) designed morphology-based filters with linear structuring elements to locate crack points.

Also, Yamaguchi and Hashimoto (2009) proposed a type of scalable percolation-based image processing method that considers crack connectivity among neighboring image pixels. The idea behind the method is simple. Suppose water is poured at one surface point. If the point is a crack point, the water will flow along the crack, and the shape is linear. If the point is a non-crack point, the water will spread evenly, and the shape is circular. This way, whether a surface point is a crack point can be determined by checking its water percolation shape (Figure 12). According to the test results from Yamaguchi and Hashimoto (2009), it indicated that the method can correctly detect cracks with efficient computation time even for a large-size concrete surface.

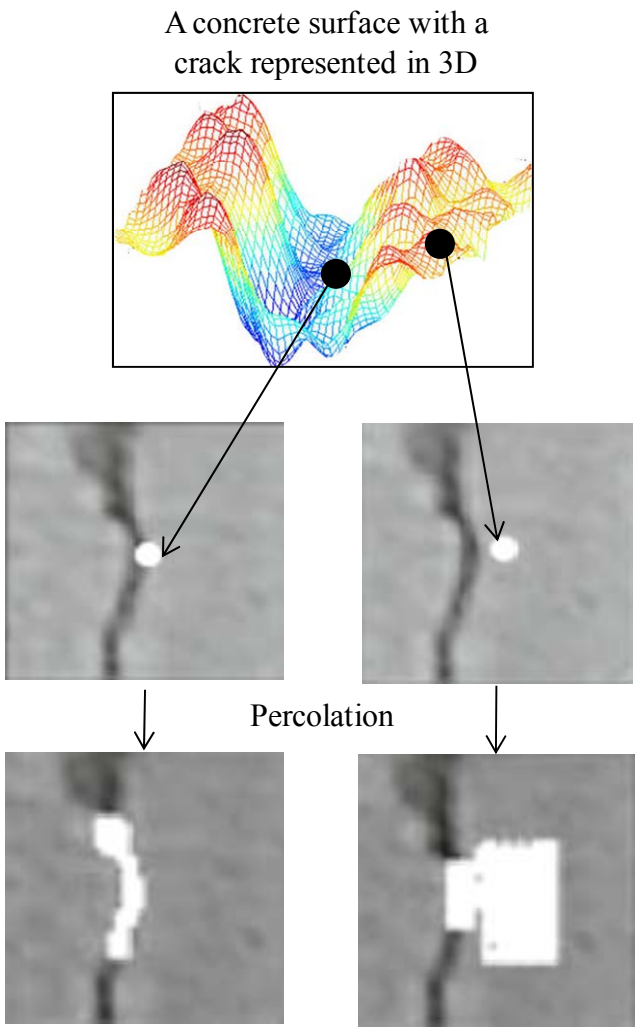


Figure 12: Percolation-based crack detection

The crack map only contains the location information of isolated crack points in a surface image. It is unknown how to organize these crack points to form different cracks directly from the crack map, and how to get the specific properties (e.g. length, orientation, and maximum width) of each crack. Little work has been focused on automatically retrieving this cracking information from the crack map. To the researcher's knowledge, Yu et al. (2007) calculated the length, thickness and orientation of concrete cracks through a graph search; however, their method required the start and end points of the crack to be manually provided first. Chae et al. (2003) relied on an

artificial neural network to retrieve crack properties, but it is unclear how to form the network's input data sets and how effective the network is.

2.4.2 Coating Rusts and Air Pockets Detection

In addition to cracks, the detection of other defects, such as coating rusts and air pockets have also been investigated by researchers. Lee et al. (2005, 2006) analyzed the visual characteristics of bridge coating images with and without rusts, and found out that three variables (MEAN in red, DIFF in green and DIFF in blue) had a significant impact in recognizing the existence of bridge coating rusts. Based on this finding, they developed an automated processor to determine whether rust defects exist in a given digital image with artificial intelligence and statistical analysis (Lee et al. 2006).

Suwwanakarn et al. (2007) proposed three circular filters to detect air pockets on the surface of concrete. One filter with a large size (11 image pixels by 11 image pixels) was used to detect large air pockets, while the other two filters with a smaller size (5 image pixels by 5 image pixels) were used to detect small air pockets. The filters with fixed size can guarantee that most detected air pockets are real air pockets. This is because, when convoluting a concrete surface image with the filters, the high responses of the filters are always expected at the places where air pockets exist and their size is similar to the size of the filters. However, there is a big limitation about the utilization of the filters with fixed size. Only air pockets with the same or similar size as the size of the filters can be accurately detected. The air pockets with a different size than the size of the filters cannot be recognized. Therefore, the method missed the detection of most real air pockets with the size different from the size of the filters (Figure 13).

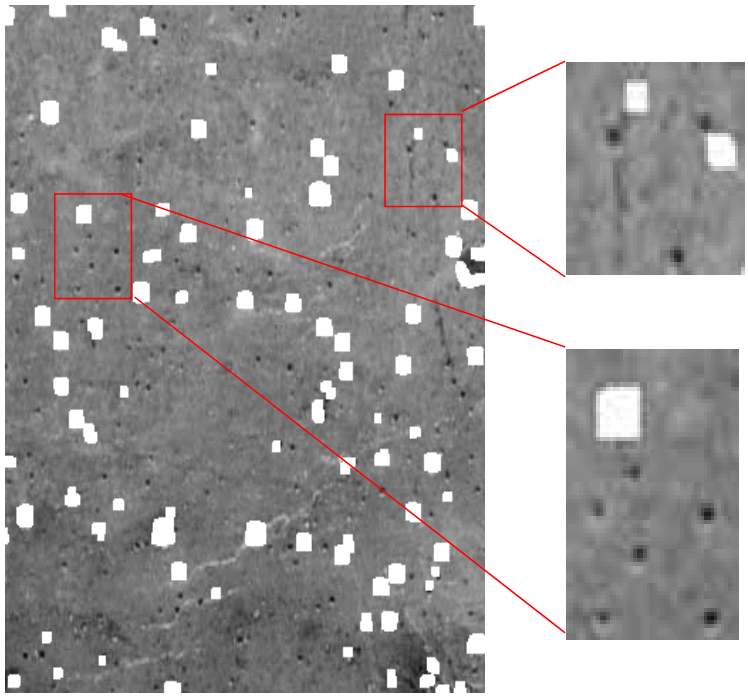


Figure 13: Undetected air pockets using the filters with fixed size

2.5 Summary

Manual visual inspection has been widely adopted in routine and emergency scenarios for inspecting civil infrastructure, such as bridges, highways, tunnels, and commercial and residential buildings. However, several limitations related to manual visual inspection have been identified, including the subjective and not always reliable nature of inspection results, the costly and time-consuming evaluation process, and the demanding requirement of experienced inspectors.

In order to overcome these limitations, recent research studies have been proposed to replicate manual visual inspection practice. Most of them are still under investigation. According to the recent development, these research studies are mainly focused on the fast and automated collection of damaging information inflicted on structures through 3D CAD models superimposition and structure health monitoring sensors. The requirements

of 3D CAD models and pre-placed sensors make them more applicable for newly developed structures rather than old, existing ones, since most existing structures that were built thirty or forty years ago were designed with 2D drawings and without the installment of any structure health monitoring sensors.

On the other hand, many damage/defects detection methods have been developed to extract defects or damage information from images and videos. The methods used image processing techniques such as thresholding, edge detection, and digital filtering, to locate cracks, air pockets and coating rusts inflicted on the surfaces of structure elements. Their effectiveness has been validated, even when well-light conditions are not available.

However, two gaps of knowledge limit their applications in inspecting structures, such as bridges, underground pipes and tunnels, either on a routine basis or in an emergency scenario. First, there was no accurate template or model that can be used to automatically locate structural elements in images and videos. Color/texture-based structural element recognition methods rely on the elements' material information to perform recognition. They cannot be used to recognize structure elements in a frame structure. The shape-based recognition methods make use of the shape information of structure elements, but the sole reliance on the shape information makes them not always reliable. Local scale/affine invariant features are powerful in describing a single specific element in multiple images, but cannot be used to detect multiple elements in one image.

Second, existing damage and defect detection methods are only limited to detecting the presence of defects or damage in the surface image of structural elements, or to locating the defects/damage points in the image. How to group the defect/damage points has not been adequately investigated. Also, the methods failed to measure the properties

of the defects and damage for the purpose of civil infrastructure assessment. The main objective of this research studies is to fill these two gaps.

CHAPTER 3
CONCRETE COLUMN RECOGNITION

This chapter describes a novel method of recognizing concrete columns in images and videos. The method overcomes the limitations of previous structural member recognition methods by combining the shape and material information of concrete columns as recognition cues. It starts from the extraction of long near-vertical lines from an image or video frame through edge detection and the Hough Transform. The bounding rectangle for each pair of lines is then constructed. When the rectangle resembles the shape of a column and the color and texture contained in the pair of lines are classified as concrete material, a concrete column surface is assumed to be located. The method has been implemented in a Visual Stuido .NET environment, and tested using real images/videos. The results were compared with manual recognition ones to validate the method.

3.1 Method Description

As mentioned in the previous chapter, a typical structural member recognition method may consider the visual appearance of the structural member in images and videos. This visual appearance includes shape, texture and color. For concrete columns, it is found that (1) the shape of one concrete column surface is dominated by a pair of long near-vertical lines, and (2) the texture and color patterns on the surface are uniform. Based on these two findings, Figure 14 illustrates the general idea of the method.

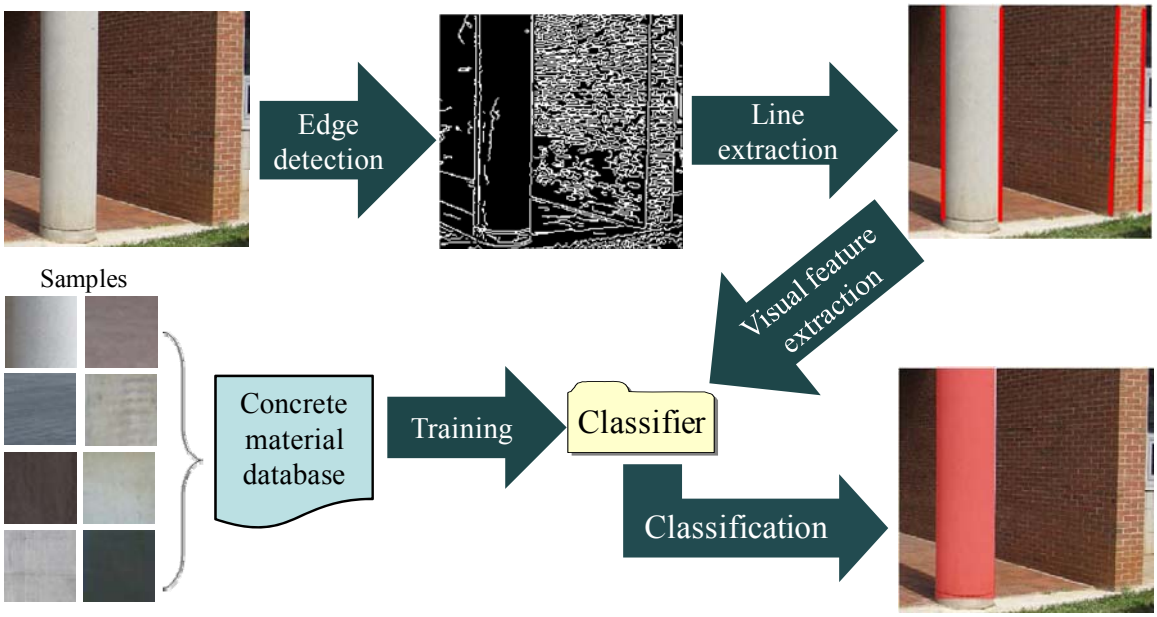


Figure 14: General idea of the method for concrete column recognition

Specifically, the edge pixels in each image or video frame are first extracted using edge detection techniques. Then, the edge pixels are grouped to form near-vertical lines. When the long near-vertical lines are identified, two neighboring long near-vertical lines are constructed as a pair, if they have similar length. The aspect ratio (width/length) of each pair is calculated. If it is larger than one, it means the distance of two lines is larger than their length, which is not common for concrete columns in a practical reinforced concrete frame structure. Therefore, such pairs are discarded. Only the pairs of long near-vertical lines that have small aspect ratios are kept, which form the potential concrete column regions in the image or video frame.

The visual features (color and texture) of each potential concrete column region are calculated and input to a concrete material classifier, which is pre-trained with positive and negative concrete samples. If the classifier determines that the material in the region is concrete, one surface of a concrete column is assumed to be located, since any concrete column in the image or video frame must have two dominant long near-vertical lines at

its sides and concrete material in the middle. In order to reduce the computation time and increate the robustness of the method, several optimizations have been made, which are described in the following sub-sections.

3.1.1 Column Boundary Detection

In the edge detection, not all possible edge pixels are kept to form near-vertical lines in an image or video frame. Instead, only near-vertical edge pixels are kept in the edge map (Figure 15). In doing so, the image or video frame is first considered as a mapping from an x-y plane to a RGB space (Red, Green, and Blue). The Jacobian matrix (J_c) of the mapping at each pixel (x, y) can be represented in Eq. 1, which indicates the color change in red, green, and blue (dR, dG, and dB) induced by moving any infinitesimal step (dx, dy) in the image x-y plane (Liu et al. 2007).

$$J_c = \begin{bmatrix} \partial R/\partial x & \partial R/\partial y \\ \partial G/\partial x & \partial G/\partial y \\ \partial B/\partial x & \partial B/\partial y \end{bmatrix} \quad \text{(Eq. 1)}$$

The Euclidean squared magnitude of the change can be calculated using Eq. 2, where the largest eigenvalue of the matrix Q is the gradient magnitude and the eigenvector corresponding to the largest eigenvalue is the gradient direction (Liu et al. 2007). When the precise gradient magnitude and direction of each pixel (x, y) are retrieved, those with local maximum gradient magnitudes in their gradient directions are marked as potential edge pixels. Near-vertical edge pixels are the pixels with near-horizontal gradient directions.

$$M^2 = (dx, dy)Q(dx, dy)^T \text{ where } Q = J_c^T J_c \quad \text{(Eq. 2)}$$

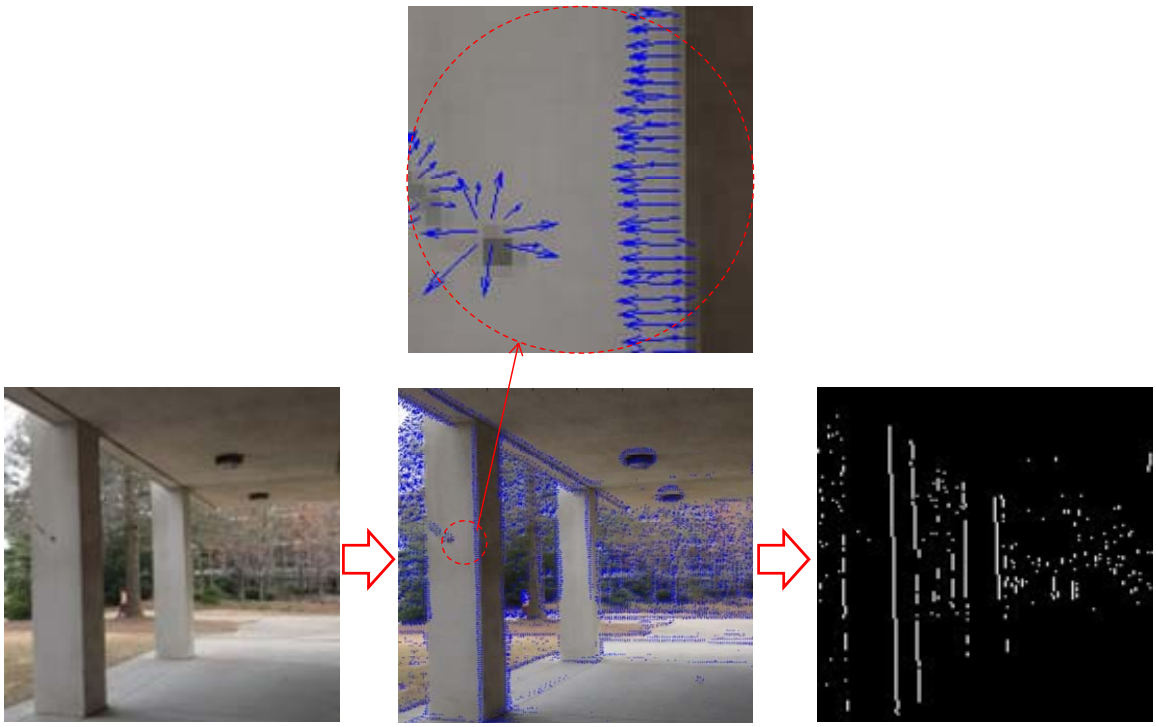
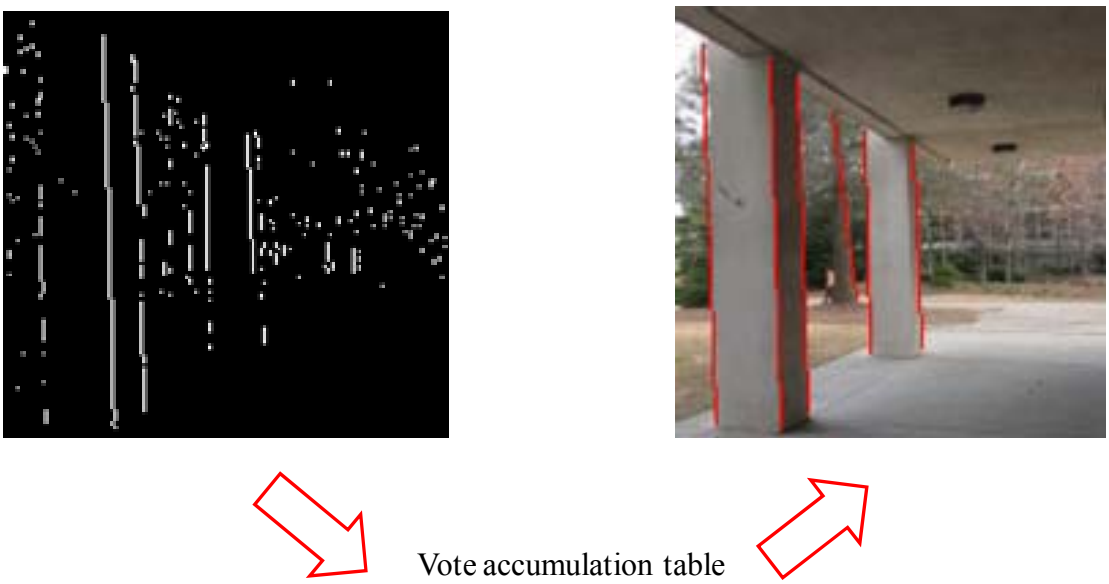


Figure 15: Near-vertical edge pixels in the edge map

Near-vertical edge pixels in the edge map are grouped to form near-vertical lines using the Hough transform (Duda and Hart, 1972). The transform maps the Cartesian coordinate (x, y) of each edge pixel in into a radius-and-angle (ρ, θ) parameterization space. In the Hough transform, the gradient orientation (θ) of the edge pixel is directly input to the Eq. 3 to calculate its corresponding ρ, instead of enumerating all possible pairs of (ρ,θ) for each (x, y).

$$\rho = x\cos(\theta) + y\sin(\theta) \qquad \text{(Eq.3)}$$

When one pair of (ρ,θ) is calculated, the value at the position of (ρ,θ) in a vote accumulation table is added by one. This means one more edge pixel supports to form the line (ρ, θ). The pairs that have high voting numbers in the table are located after all near-vertical edge pixels are explored. The corresponding pairs of (ρ,θ) can be then used to formulate long near-vertical lines (Figure 16).



Vote accumulation table

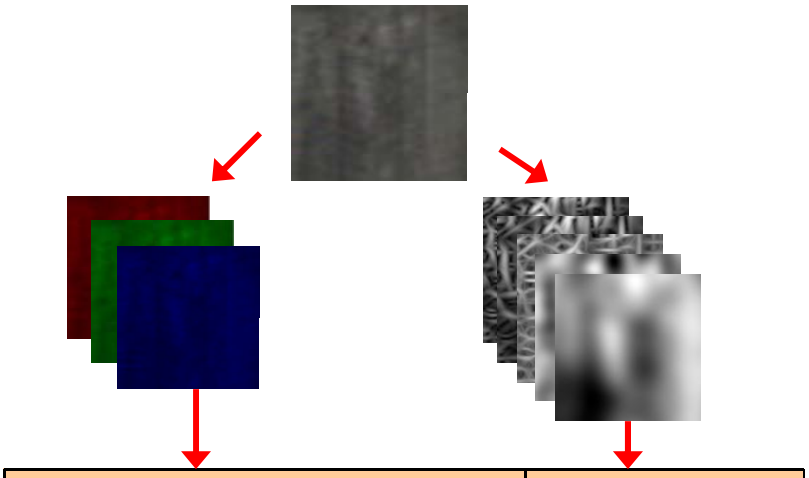
θ	ρ				
	-2	-1	0	1	2
-18°	5				
-9°	52	16	8		
0°			10		
9°				15	
18°					

Figure 16: Near-vertical lines extraction

After all long near-vertical lines are retrieved, each of them is compared with its neighboring ones. If two neighboring lines have a similar length, they are regarded as a pair. The comparison is performed iteratively, till no line can find its partner to form a pair. The distance between each pair of vertical lines is calculated as the width of the pair. The aspect ratio (width/length) of the pair is future measured. If the ratio is larger than one, it means the width of the pair is larger than its length. This is not common for concrete columns in a practical reinforced concrete frame structure. Therefore, those pairs are removed. Only the pairs of long near-vertical lines that have small aspect ratios are kept to form the boundaries of candidate concrete columns.

3.1.2 Concrete Material Classification

When the boundaries of candidate concrete columns are identified, the visual features of the region in each boundary are calculated. The features in this research study include two parts: color and texture (Figure 17). Color information (red, green and blue) is represented by two ratios. The first one is the ratio of red to green (R2G) and the other is the ratio of blue to green (B2G). This way, the brightness effect can be removed from the color information. Texture information is retrieved using the RFS filter bank, where only the maximum filter response across all orientations and scales are recorded to ensure the texture information is scale and rotational invariance (Varmaand Zisserman, 2005).



Color				Texture	
R2G		B2G		Edge	Bar
Mean	STD	Mean	STD	Max	Max
1.001	0.016	0.958	0.026	0.393	0.155

Figure 17: Example of visuale features

Figure 18 shows the visual features of four samples (concrete 1, concrete 2, wood and sky). It can be found that the combination of color and texture can visually characterize concrete material in an image. For example, both concrete and sky show similar color characteristics (two color ratios are close to one), but concrete has higher response values

to the bank of the filters than sky has. In contrast, although concrete and wood have high response values to the bank of the filters, their color characteristics change significantly.

Concrete 1 Concrete 2 Wood Sky

Samples	Color				Texture	
	R2G		B2G		Edge	Bar
	Mean	STD	Mean	STD	Max	Max
Concrete 1	1.009	0.007	0.982	0.007	0.505	0.141
Concrete 2	0.880	0.011	1.099	0.017	1.677	0.952
Wood	1.239	0.056	0.736	0.059	1.185	0.645
Sky	0.918	0.008	1.014	0.006	0.097	0.057

Figure 18: Visual features for concrete, wood, and sky

The concrete material classification boundaries in the feature space are determined using machine learning techniques (Figure 19). Two types of machine learning techniques (SVM and ANN) are investigated in this research study. One challenge of creating machine learning based classifiers is to determine their parameters. Here, the parameters, such as the SVM kernel type, the number of ANN layers, and the number of neuron-nodes per layer are appropriately determined in the reference to previous work as well as the experimental results performed later.

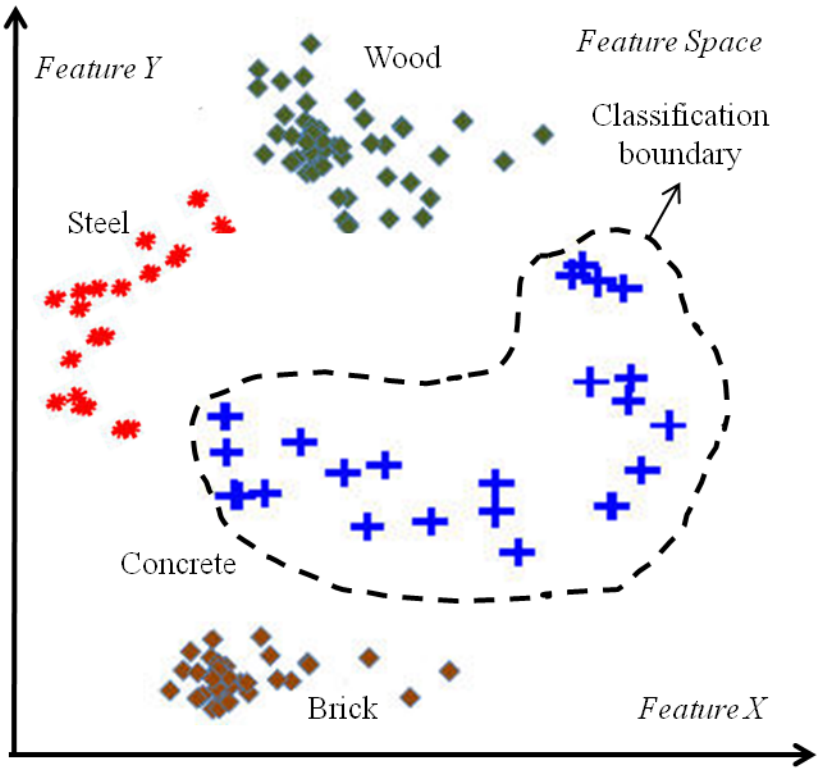


Figure 19: Concrete material classificaiton boundaries

In general, the SVM-based classifier is equipped with a radial basis function, as recommended by Hsu et al. (2010), to map non-linear visual features into a high dimensional feature space. The ANN-based classifier has the sigmod function on its neuron-nodes. The classifier is composed of three layers (one input layer, one hidden layer and one output layer), although other different multilayer solutions also have the equivalent classification functions. The number of neuron-nodes in the input layer equals to the number of visual features, while the number of neuron-nodes in the hidden layer is determined experimentally.

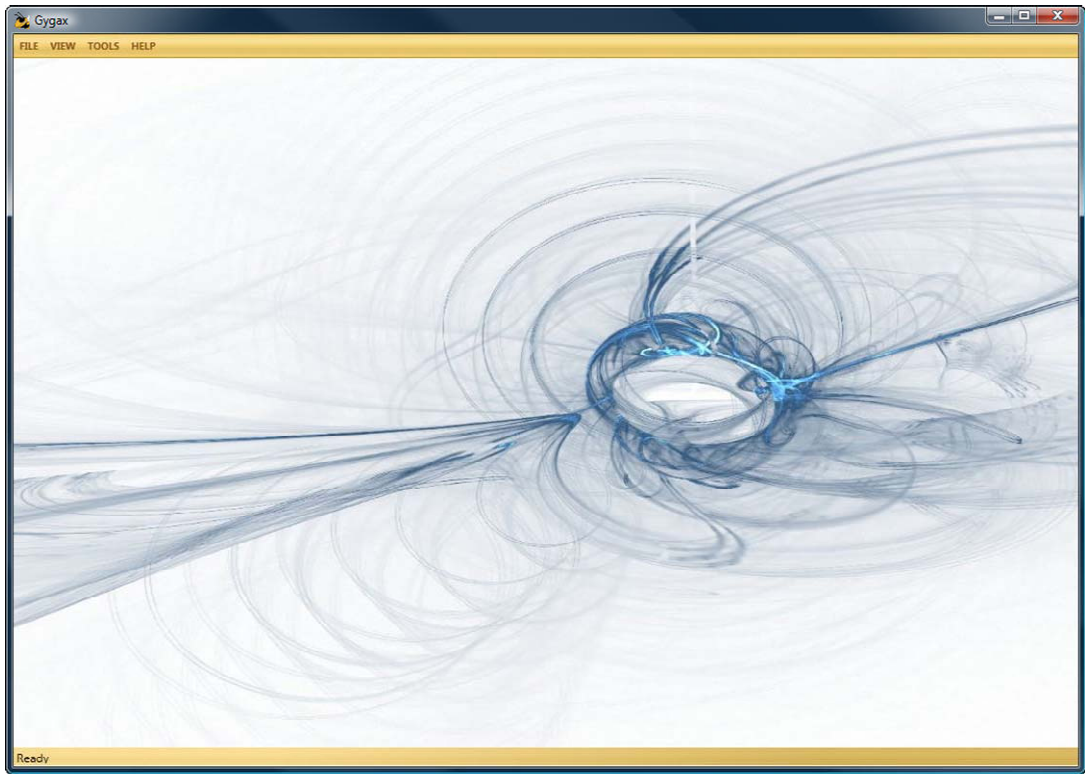
Both SVM-based and ANN-based classifiers need to be pre-trained. The training can be repeated up to thousands of times. Although it is possible to have more repetitions to further reduce the error rates on the classification of training samples, the long training to

low errors results in an overtraining problem. The classifiers only memorize the peculiar training patterns of the training samples, but fail to generalize them. When the training completes, the classifiers can be used to determine whether the input visual features belong to concrete. If concrete, then one concrete column surface is located.

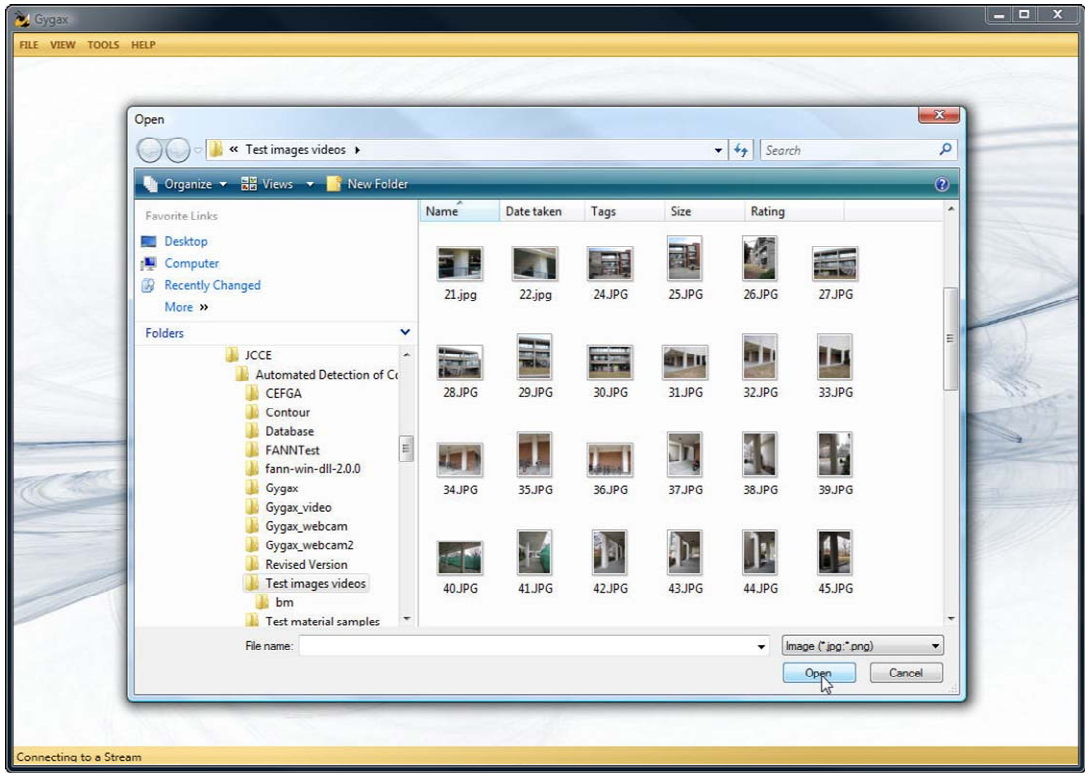
3.2 Implementation

The work of automated concrete column detection was implemented and integrated into the prototype that was developed by the Construction Information Technology Laboratory at the Georgia Institute of Technology as an independent module. The prototype was written in Microsoft Visual Studio .NET, and it was tested by the authors to collect the images and videos of structures that were damaged in the January Haiti earthquake. Intel® Open Source Computer Vision Library (OpenCV) was used as the prototype's main image processing toolbox, and EmguCV was used as a wrapper to allow OpenCV functions to be called in the prototype. Both OpenCV and EmguCV are open source (Bradski and Kaehler, 2008; Egmu CV, 2010).

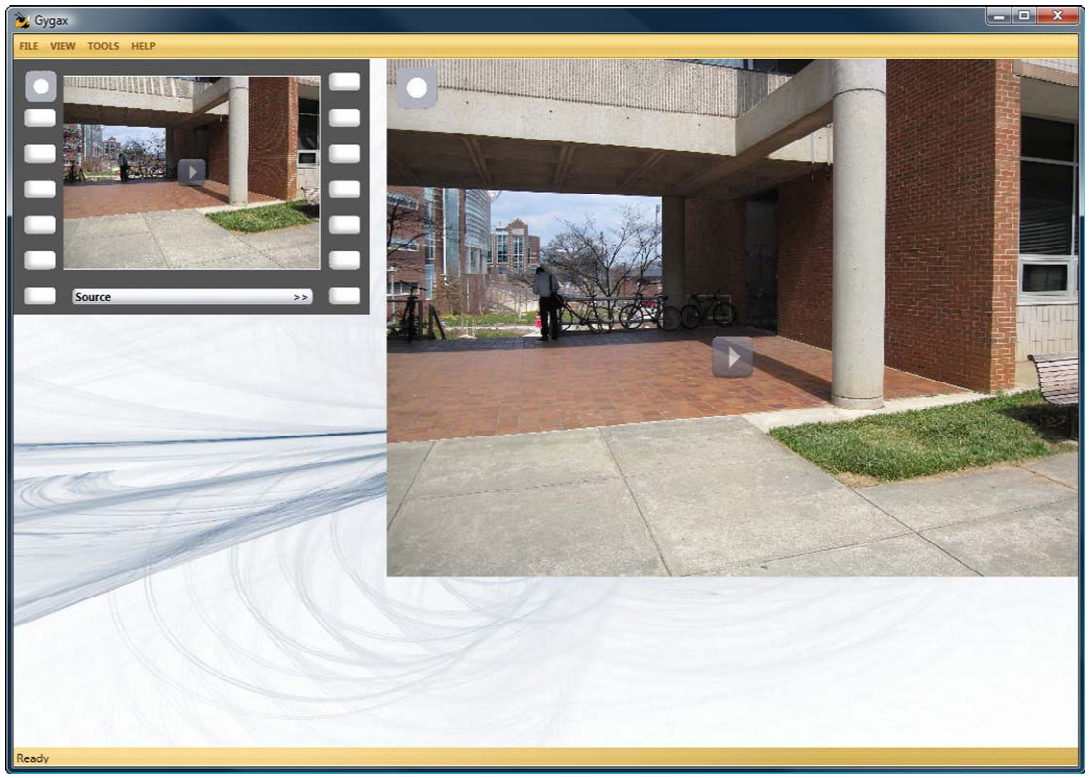
Figure 20 shows the screenshots of using the prototype to recognize concrete columns in images. Figure 20 (a) is the main interface of the prototype. A user can browse the folders (Figure 20 (b)) to load an image or video into the prototype (Figure 20 (c)). When the user selects the "process" option in the pop-up menu of the prototype (Figure 20 (d)), a concrete column in the image is recognized and marked red (Figure 20 (e)).



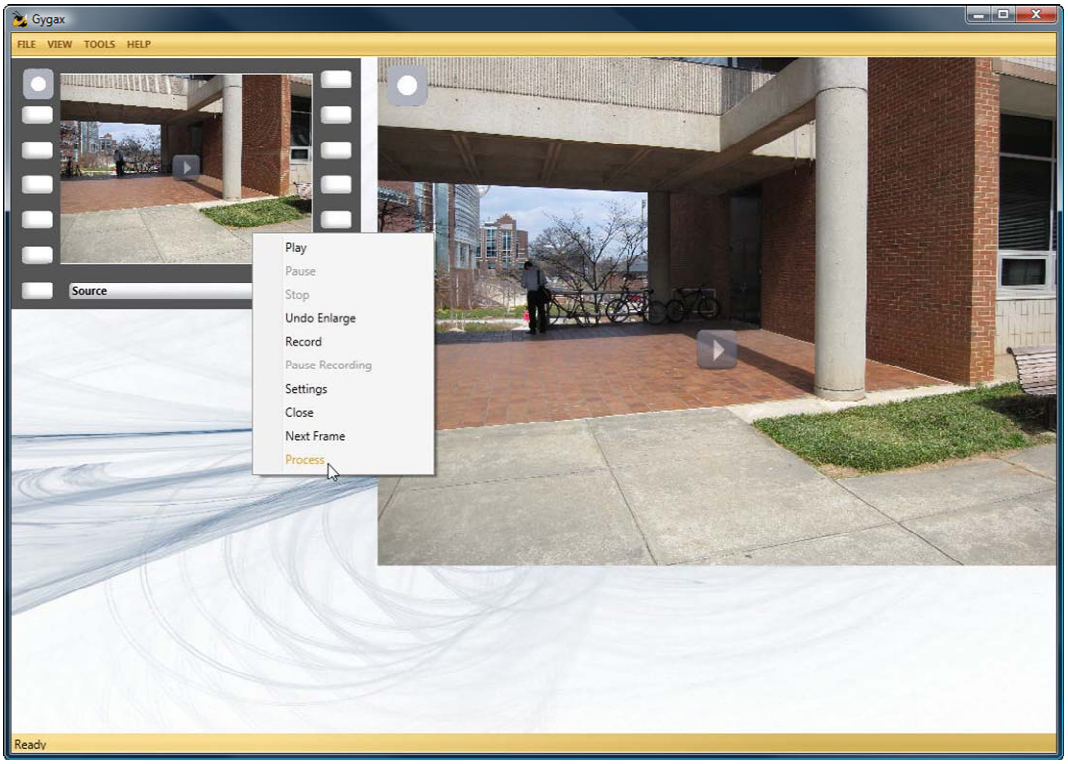
(a) Main interface of the prototype



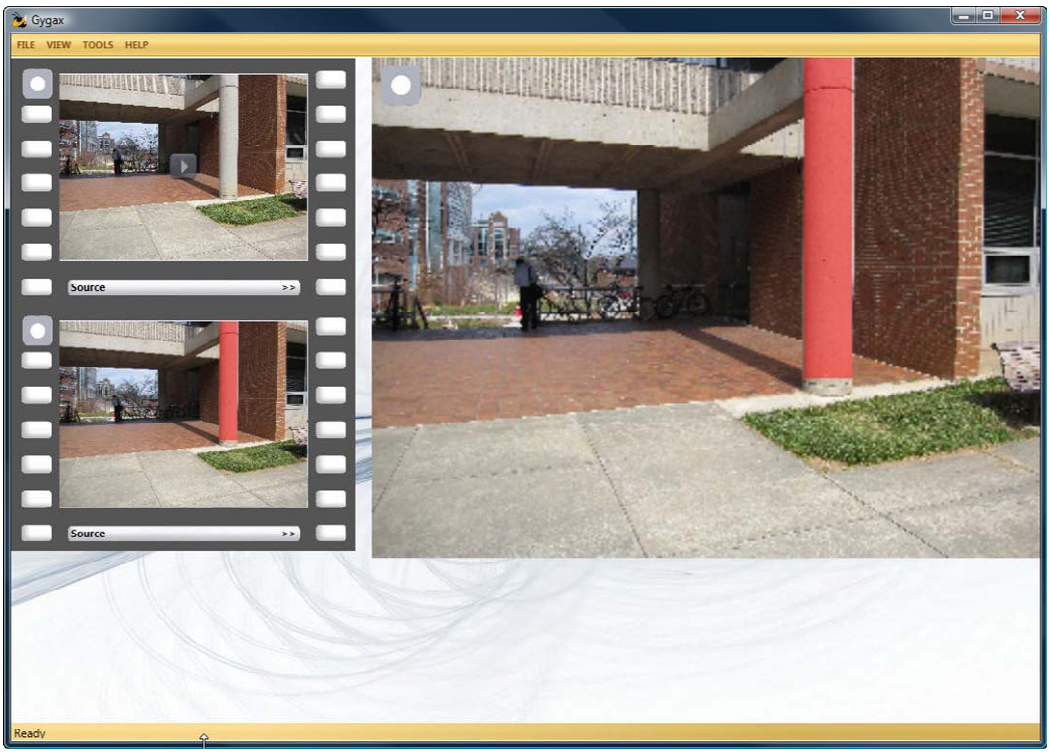
(b) Image browsing



(c) Image loading



(d) Image processing



(e) Recognition result

Figure 20: Prototype of concrete column recognition

A simplified prototype was also developed to recognize concrete columns in the videos captured by a web camera (Logitech Pro 9000) in real time (Figure 21). The web camera can be mounted on the evaluator's hardhat and connected to a Laptop (HP Compaq 2210b with Intel® Core™2 Duo Processor T7100 and 1G Memory) through the laptop's Universal Serial Business (USB) interface. When the resolution of the video frames captured by the web camera was set up at 320 by 240, the speed of concrete column recognition can reach 25 fps (frames per second).

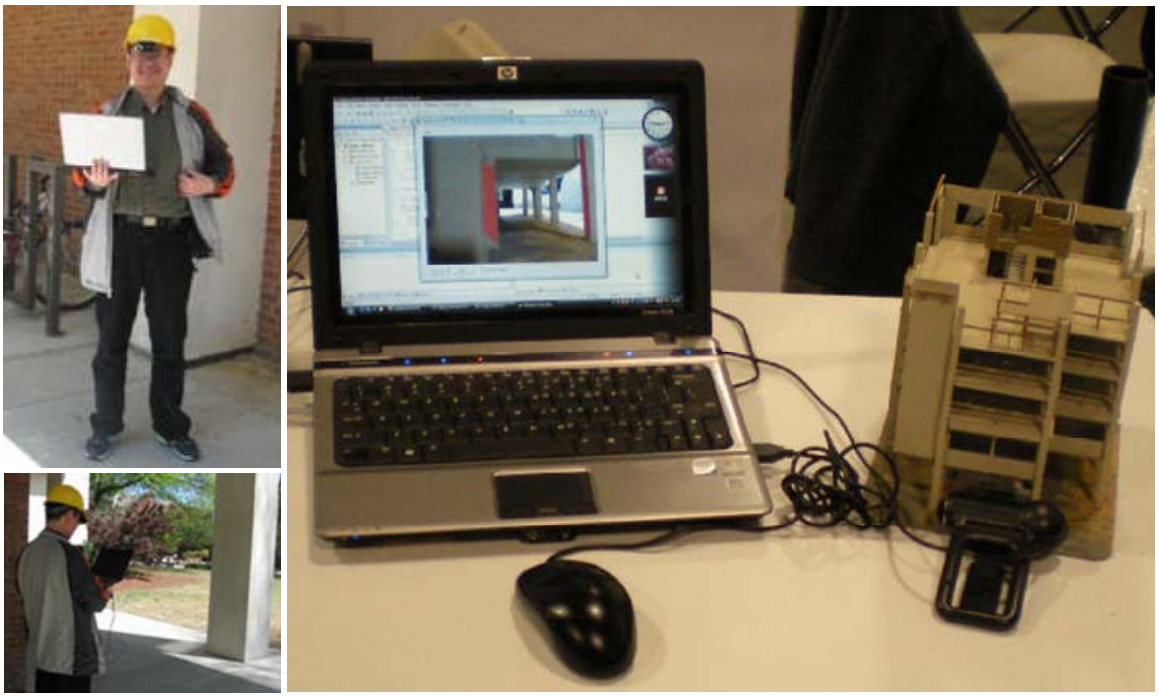


Figure 21: Concrete column recognition in real time

3.3 Results

3.3.1 Training and Testing Samples for Concrete Material Classifiers

A set of training samples for the training of the classifiers include 63 positive concrete samples and 51 negative concrete samples. Negative concrete samples are considered when performing binary classification. The set used for the validation of the classifiers includes 167 samples (53 positive concrete samples and 114 negative concrete samples). Figure 22 shows some examples of positive and negative concrete samples that were used for training and testing the concrete material classifiers using the following machine learning techniques: Support Vector Data Description, C-Support Vector Classification, and Artificial Neural Network.

Negative concrete samples

Positive concrete samples

Figure 22: Examples of training and testing samples

3.3.2 Support Vector Data Description (SVDD) Classifier

The SVDD classifier is a type of one-class classifiers. The classification boundaries determined by the SVDD classifier are based on the distribution of positive concrete samples in the feature space only. The parameter of the SVDD classifier is gamma, which controls the width of the Radial Basis Function (RBF) kernel in the SVDD classifier. Large gamma makes the kernel value go to zero quickly, which reproduces the irregular decision boundaries from the training samples. In contrast, small gamma makes the kernel value smooth, and therefore avoids reproducing noise in the training samples.

In order to investigate the gamma effectiveness on the classification accuracy of the training and testing samples, different gamma values are selected from 0.1 to 40. The training accuracy (the number of correctly classified training samples over the number of

training samples) and the test accuracy (the number of correctly classified samples over the number of test samples) for each gamma value are illustrated in Figure 23. Overall, both training and test accuracy are initially improved with the increase of the gamma values. This trend is kept, until the gamma reaches 6. After that, the increase of the gamma values makes both training and test accuracy reduce.

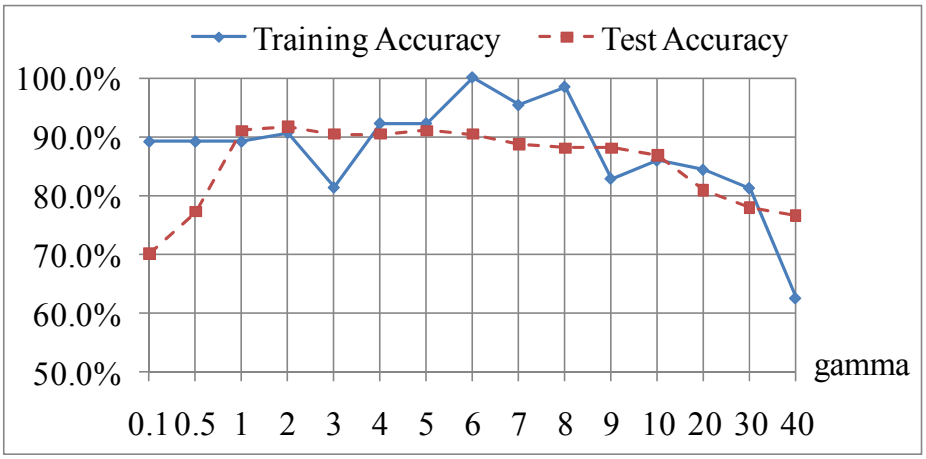


Figure 23: Training and test accuracy for the SVDD classifier at different gamma values

3.3.3 C-Support Vector Classifier (C-SVC)

The C-SVC is a type of two-class classifiers. Other types of two-class classifiers, such as nu-SVC, are almost equivalent but quipped with different parameters. The C-SVC with the RBF kernel is determined by two parameters: C (the cost coefficient) and gamma. The gamma in the C-SVC has the same function as the gamma in the SVDD classifier, while C controls the fitting degree of the classifier. Small C always makes the classifier under-fitting, while large C produces the over-fitting problem.

In order to find an optimum combination of C and gamma for the C-SVC, a grid search is adopted, where the C is enumerated from 16 to 8192, and the gamma is enumerated from 0.25 to 32. Figure 24 shows the effectiveness of the C on the training

and test accuracy of the samples with a fixed gamma. It can be found that the training accuracy is gradually improved to 100% with the increase of the C, however, the test accuracy stops growing after C reaches 512. This is because the classifier experienced the switch from the under-fitting to the over-fitting. Figure 25 shows the effectiveness of the gamma on the training and test accuracy of the samples with a fixed C. In Figure 25, it can be found that training accuracy grows till 100% with the increase of the gamma, but the test accuracy stops growing and starts to drop after the gamma reaches 2.

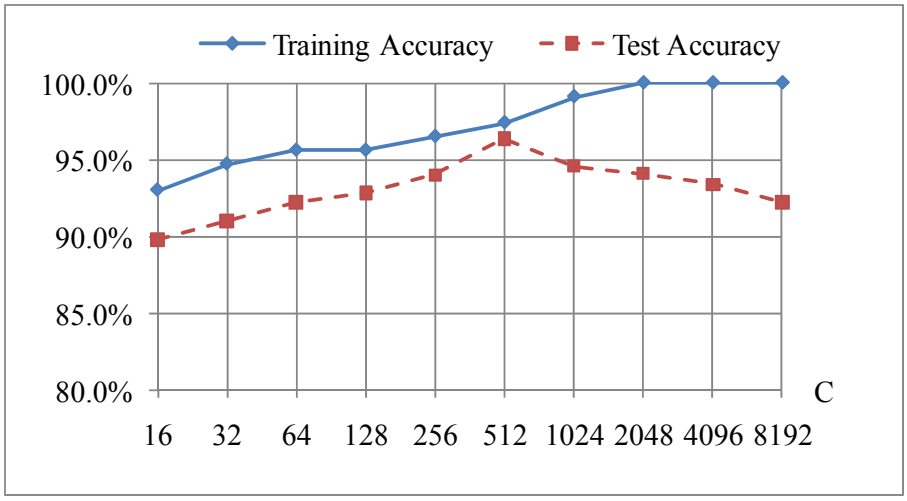


Figure 24: Training and test accuracy for the C-SVC at different C values

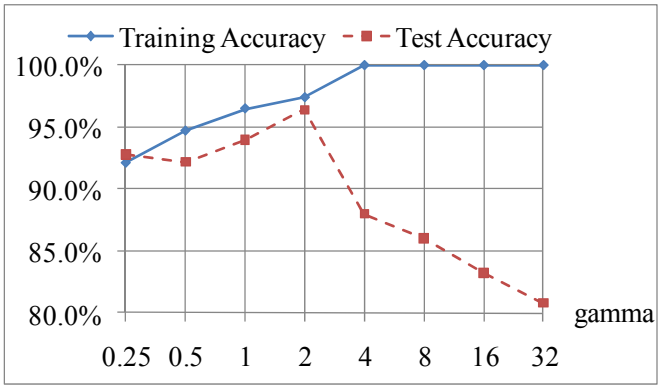


Figure 25: Training and test accuracy for the C-SVC at different gamma values

3.3.4 Artificial Neural Network (ANN) Classifier

In the ANN-based classifier, the number of network layers (three) and the transfer function at each node (the sigmod function) are pre-determined. Other parameters, such as the number of the neuron-nodes in the hidden layer and the number of the training repetitions, are determined experimentally. A grid search for these parameters is performed. Specifically, the classifier is configured with the different number of neuron-nodes in the hidden layer from 2 to 19, and the training is repeated from 1,000 times to 300,000 times.

Figure 26 shows the effectiveness of the number of the neuron-nodes in the hidden layer. It can be seen that the training accuracy is improved with the increase of the number of neuron-nodes initially, and then the training accuracy is waving around 99% when the number of neuron-nodes is larger than 7. The test accuracy reaches the maximum when the number of neuron-nodes is 18.

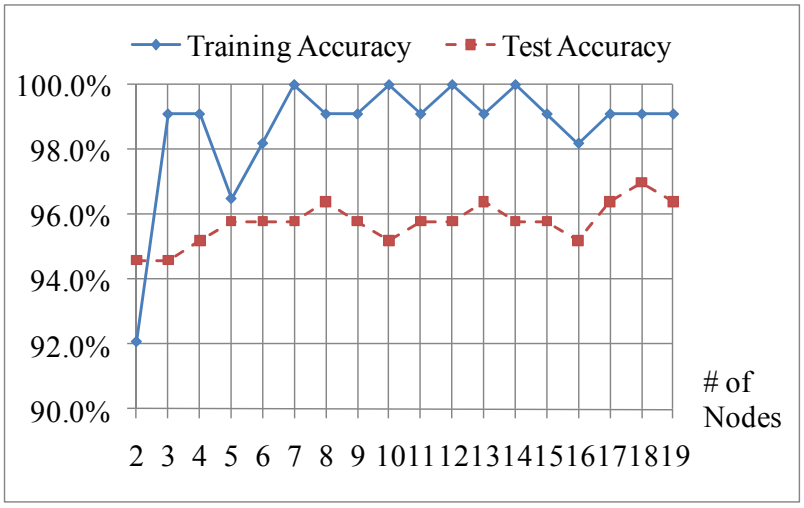


Figure 26: Training and test accuracy for the ANN classifier at different number of neuron-nodes in the hidden layer

Figure 27 shows the effectiveness of the training repetitions on the training and test accuracy. The training accuracy gradually increases with the increase of the repetitions. However, the test accuracy increases after reaching a certain limit (9,000), and then drops significantly.

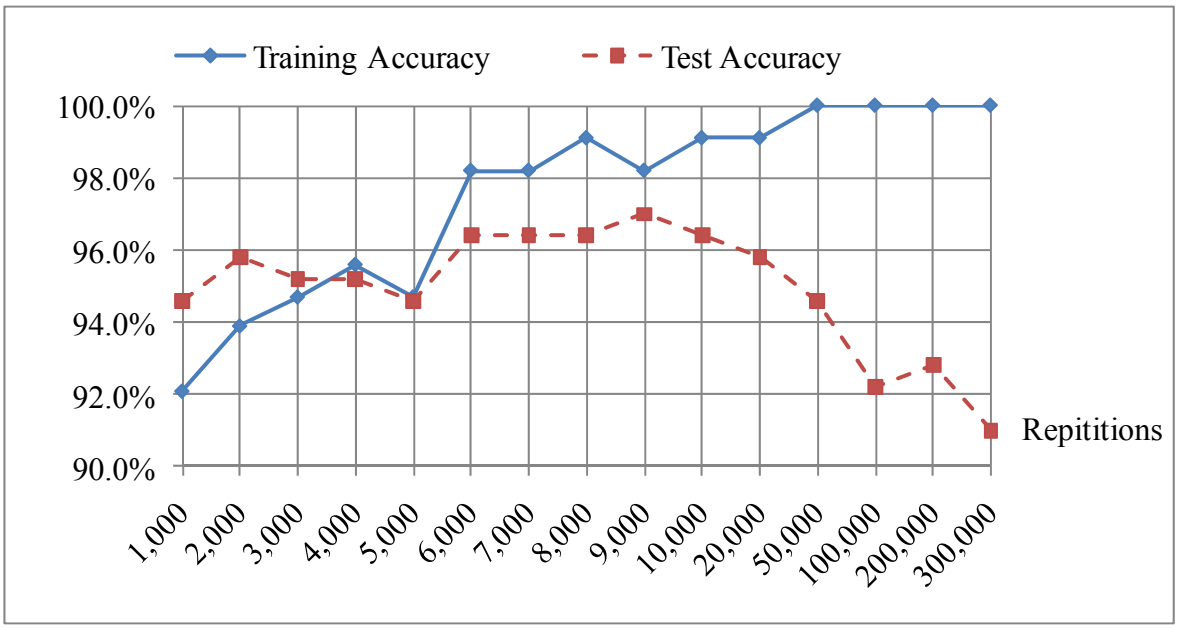


Figure 27: Training and test accuracy for the ANN classifier at different training repetitions

3.3.5 Concrete Material Classification

The ANN classifier is adopted based on the comparison of previous three classifiers. A database of real construction site images was used to test the performance of the ANN classifier in classifying concrete regions in the images. The images were taken at different construction sites under normal natural light conditions. Figure 28 is one example of classifying concrete regions. In doing so, the image is first divided into a set of areas using existing image segmentation techniques. Then, the visual features of each

area are calculated and input to the classifier. For the areas that are determined as concrete areas by the classifier, they are marked red.

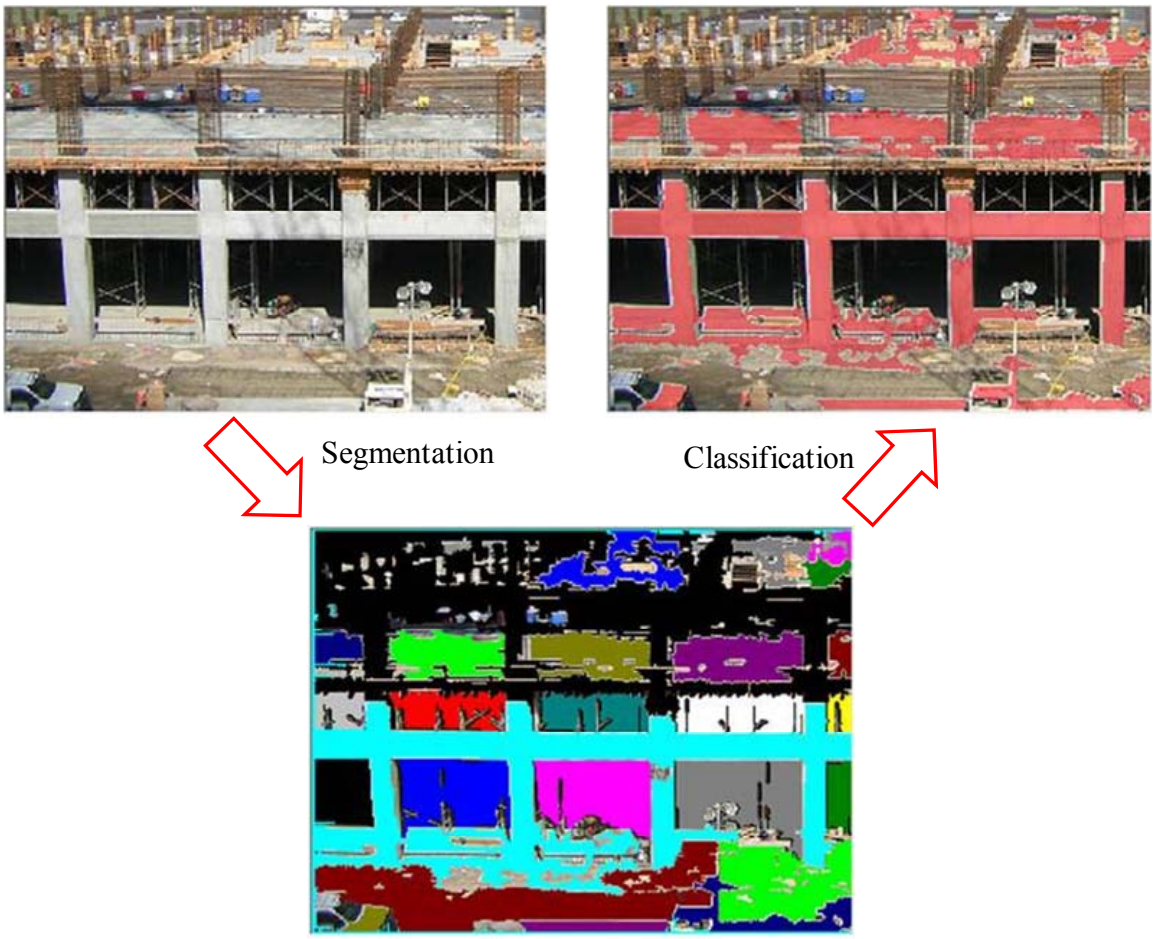


Figure 28: Example of classifying concrete regions

The performance of the classifier in classifying concrete regions is measured by two ratios: 1) Precision and 2) Recall. The precision here is defined as the percentage of the concrete areas correctly classified in the total concrete areas classified. The recall ratio is the percentage of the concrete areas correctly classified in the real concrete areas. The real concrete areas in an image are manually identified. Considering the difficulty of measuring the irregular concrete areas in the image, the number of image pixels in each area is counted to approximate the area size. This way, the precision equals to the number

of correctly classified concrete pixels divided by the number of classified concrete pixels, and the recall equals to the number of correctly classified concrete pixels divided by the number of real concrete pixels.

Table 1 records the number of correctly classified concrete pixels, the number of classified concrete pixels, the number of real concrete pixels, and the total number of image pixels for ten images. The number of real concrete pixels and the number of correctly classified concrete pixels in each image are counted as follows. First a special color (e.g. pure green) is used to manually mark real concrete areas in an image. Then, a program is developed to read the image and identify the image pixels manually marked, based on their special color. Compared with the locations of the concrete pixels manually marked and the concrete pixels classified by the classifier, the concrete pixels correctly classified in the image are identified.

Table 1: Classified concrete and non-concrete pixels

No.	# of correctly classified concrete pixels	# of classified concrete pixels	# of real concrete pixels	# of total image pixels
1	20988	22613	23083	599130
2	8185	9833	10851	494760
3	10172	11619	11219	520557
4	26119	35249	33134	363302
5	11511	13069	11914	498498
6	22431	25143	30559	661004
7	33739	40248	48971	660676
8	16322	19896	31758	607308
9	14670	17060	17861	316092
10	14078	21183	16135	227098

Based on the classified pixels in Table 1, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) of the classification are retrieved. TP is the

number of the concrete pixels correctly classified; TN is the number of the non-concrete pixels correctly classified; FP is the number of the concrete pixels incorrectly classified; and FN is the number of the real concrete pixels that are not classified. Moreover, the precision, recall, and generality are further calculated. The precision is calculated as TP / (TP+FP); the recall is TP / (TP+FN); and the generality is (TP+FN) / (TP+TN+FP+FN). Table 2 illustrates all the measurements. It can be seen that the precision ratios vary from 66.5% to 92.8% and the recall ratios are from 51.4% to 96.6% for the ten images. The average precision and recall reach 83.3% and 79.6% separately.

Table 2: Classification performance

No.	True Positive (TP)	True Negative (TN)	False Positive (FP)	False Negative (FN)	Precision	Recall	Generality
1	20988	574422	1625	2095	92.8%	90.9%	3.9%
2	8185	482261	1648	2666	83.2%	75.4%	2.2%
3	10172	507891	1447	1047	87.5%	90.7%	2.2%
4	26119	321038	9130	7015	74.1%	78.8%	9.1%
5	11511	485026	1558	403	88.1%	96.6%	2.4%
6	22431	627733	2712	8128	89.2%	73.4%	4.6%
7	33739	605196	6509	15232	83.8%	68.9%	7.4%
8	16322	571976	3574	15436	82.0%	51.4%	5.2%
9	14670	295841	2390	3191	86.0%	82.1%	5.7%
10	14078	203858	7105	2057	66.5%	87.3%	7.1%

Precision = TP/(TP+FP); Recall = TP/(TP+FN); Generality = (TP+FN)/(TP+TN+FP+FN)

3.3.6 Concrete Column Recognition

The data set that was used to test the performance of concrete column recognition in this research study includes hundreds of images and videos from three parts. The first part is the images and videos of structures in Atlanta and Ann-Arbor. The second part is the images and videos of the structures damaged by the 2010 Haiti earthquake. The last

part is the images and videos of a parking garage in the midtown of Atlanta, which collapsed in the summer of 2009. All the videos were decomposed into a series of video frames for the test purpose.

Figure 29 shows examples of the recognition of undamaged concrete columns, while Figure 30 shows examples of the recognition of damaged concrete columns. The tested concrete columns include building columns, bridge columns, and the columns in parking garages. It is worth mentioning that all the images in Figure 29 and Figure 30 were captured by digital cameras at different locations (indoors vs. outdoors) under various light conditions (natural vs. artificial).

Figure 29: Recognition of concrete columns in undamaged structures

Figure 30: Recognition of concrete columns in damaged structures

In civil infrastructure inspection and assessment, a concrete column may be large. It is difficult to capture the column and its detailed defects and damage information with a single shot, due to the limited field of view (FOV) of a typical compact camera. Therefore, multiple images have to be taken to capture the column's different segments. The methods proposed in this research study can still be used to recognize this large-scale

concrete column by stitching these images together. Figure 31 shows an example of recognizing a large-scale bridge concrete column captured by two images.

Figure 31: Recognition of a large-scale concrete column using image stitching

In order to measure the performance of concrete column recognition, the precision and recall are calculated based on the criteria in signal detection theory (Wickens, 2002). Here, the test results were compared with manual recognition results to find the True Positive (TP), False Positive (FP), and False Negative (FN) of the results. Precision is calculated as TP / (TP + FP), where TP is the number of correctly recognized concrete columns, FP is the number of incorrectly recognized concrete columns, and (TP+FP) is the total number of recognized concrete columns. Precision measures the recognition exactness or fidelity. High recognition precision indicates that many recognized concrete columns are real concrete columns; whereas low recognition precision indicates that few recognized concrete columns are real concrete columns. Recall is calculated as TP / (TP+FN), where FN is the number of real concrete columns that are not recognized and (TP+FN) is the number of real concrete columns. Recall measures the recognition completeness. High recognition recall implies that many real concrete columns are correctly recognized; whereas low recall implies that few real concrete columns are

correctly recognized. Both precision and recall represent the quality of the recognition results retrieved by the method in this research study.

The designed experiments begin with the manual recognition of real concrete columns in each image first. Then, the method in this research study is used to recognize concrete columns which were marked in red. This way, the correctly recognized concrete columns can be identified to determine the number of correctly recognized concrete columns (TP), the number of recognized concrete columns (TP+FP), and the number of real concrete columns (TP+FN). This information is used to calculate the precision and recall.

Table 3 illustrates that the average recognition precision of the method is 89.1% and recall is 79.1%. The precision and recall for each single example are not used because they do not truly reflect the overall performance of the method. In most test images and videos, concrete columns were taken closely. Only one or two columns are visible. As a result, the prevision and recall ratios vary significantly case by case. For example, suppose there are two visible concrete columns in an image. If both columns are recognized, the precision and recall for the image are 100% and 100%. However, if one column is not correctly recognized, the precision will drop to 50%. Similarly, if one column fails to be recognized at all, the recall will drop to 50%.

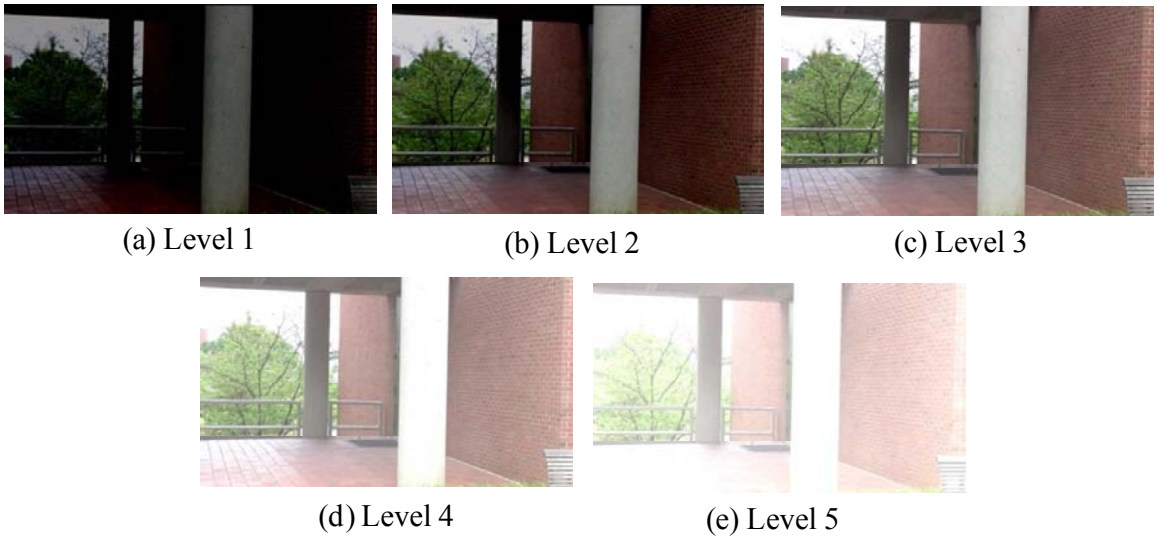
Table 3: Precision and Recall for Concrete Column Recognition

Summarization	
# of correctly recognized concrete columns (TP):	336
# of incorrectly recognized concrete columns (FP):	41
# of real concrete columns not recognized (FN):	89
Average precision (TP/(TP+FP)):	89.1%
Average recall (TP/(TP+FN)):	79.1%

3.4 Benefits and Limitations

The images and videos captured by cameras may not always have ideal qualities due to factors such as variable limited lighting, camera jittering, and out-of-focus shooting. In order to test the accuracy of the approach for recognizing concrete columns in these conditions, five metrics have been proposed, including 1) illumination, 2) jittering, 3) blurring, 4) occlusion, and 5) zooming. The performance of the approach has been measured by referring to the metrics.

Illumination is related to lighting conditions. Overly bright or weak lighting makes the boundaries of concrete columns unclear and difficult to identify. Also, it changes the columns' material appearances at their surfaces. In order to measure the effectiveness of illumination in column recognition, five illumination levels are produced for each image or video frame by increasing or decreasing its intensities as recommended by Park et al. (2011). Examples of concrete columns under different illumination levels are illustrated in Figure 32.



(a) Level 1 (b) Level 2 (c) Level 3

(d) Level 4 (e) Level 5

Figure 32: Concrete columns under five illumination levels

The average precision and recall for recognizing concrete columns in the images or video frames under different illumination levels have been recorded and illustrated in Figure 33. According to the test results, it can be seen that overly bright lighting reduces both column recognition precision and recall. Most concrete columns in the overly bright lighting conditions cannot be recognized at all. Meanwhile, some non-concrete columns are mistakenly recognized as concrete columns by the approach. In the case of weak lighting, the number of real concrete columns that are recognized by the approach also drops, which results in low recognition recall. However, if recognition is made by the approach, the recognized concrete columns are real concrete columns in most cases. This leads to high recognition precision.

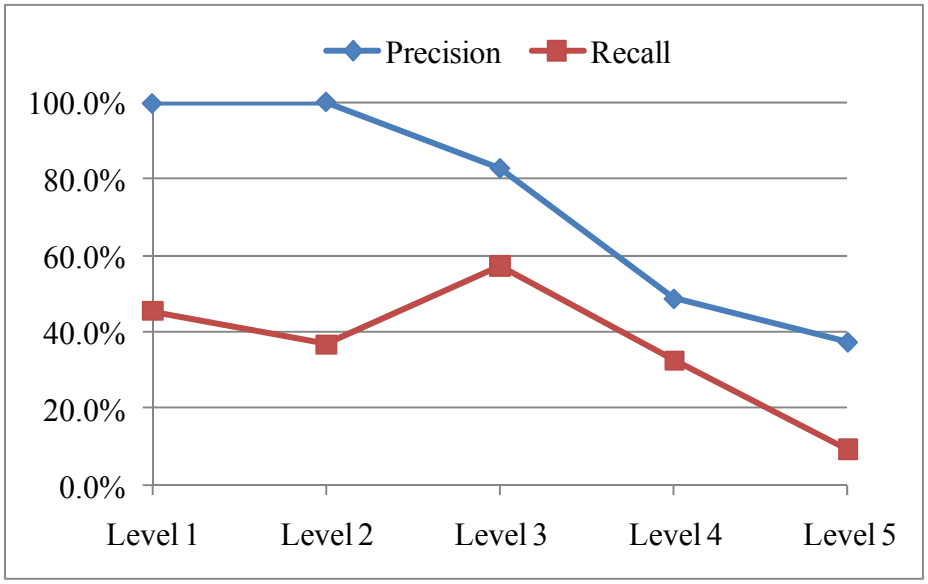
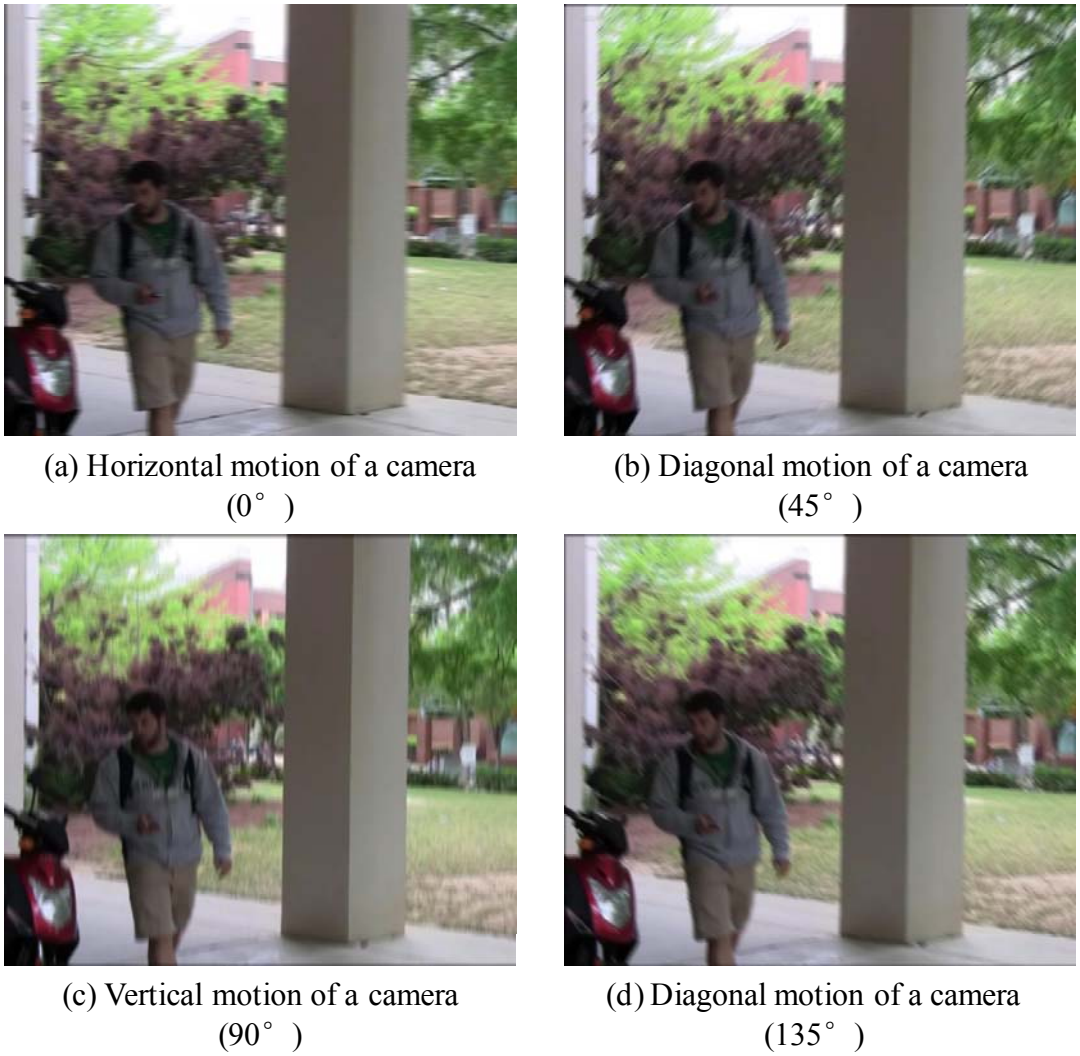


Figure 33: Recognition precision and recall for concrete columns under different illumination levels

When capturing images or videos, concrete columns in a scene may be inflicted with a motion blur due to a camera jitter or shake. This motion blur is not only limited to video cameras, which cause visible frame-to-frame jitters in their recorded videos. Also, it

happens on still cameras set with slow shutter speed, since a picture represents an integration of all parts of the scene over the exposure period determined by the shutter speed. As a result, the jitter or shake of a camera with respect to concrete columns in a scene makes the columns look blurred along the relative camera movement direction. In order to approximate the effect of a camera jitter or shake, 16 digital filters are designed by considering the shifts of 2, 4, 10, and 20 pixels in an image or video frame at horizontal (0°), veridical (90°), and diagonal (45° and 135°) directions. Figure 34 shows the examples of an image imposed with the linear camera motion at four directions.



(a) Horizontal motion of a camera (0°)

(b) Diagonal motion of a camera (45°)

(c) Vertical motion of a camera (90°)

(d) Diagonal motion of a camera (135°)

Figure 34: Columns with linear camera motion at four directions (0°, 45°, 90°, and 135°)

The average precision and recall for recognizing concrete columns in the images and video frames inflicted with different levels of camera shifts are illustrated in Figure 35, 36, 37, and 38 separately. In general, it can be seen from the test results that both precision and recall ratios for concrete column recognition drop, when camera jitters or shakes increase. The more the pixels are shifted, the worse the precision and recall become. This is especially true for the diagonal shifts (45° and 135°). Both precision and recall ratios drop significantly, when the number of the shifted pixels is increased from 4 to 10. In the vertical shift (90°), both precision and recall also drop with the increase of the shifted pixels, but the drop rates are smooth compared with the drop rates in the diagonal shifts. As for the horizontal shift (0°), the test results have shown that the approach is not very sensitive to the horizontal linear motion of a camera. The precision and recall ratios are stable and maintained around 80% and 40%.

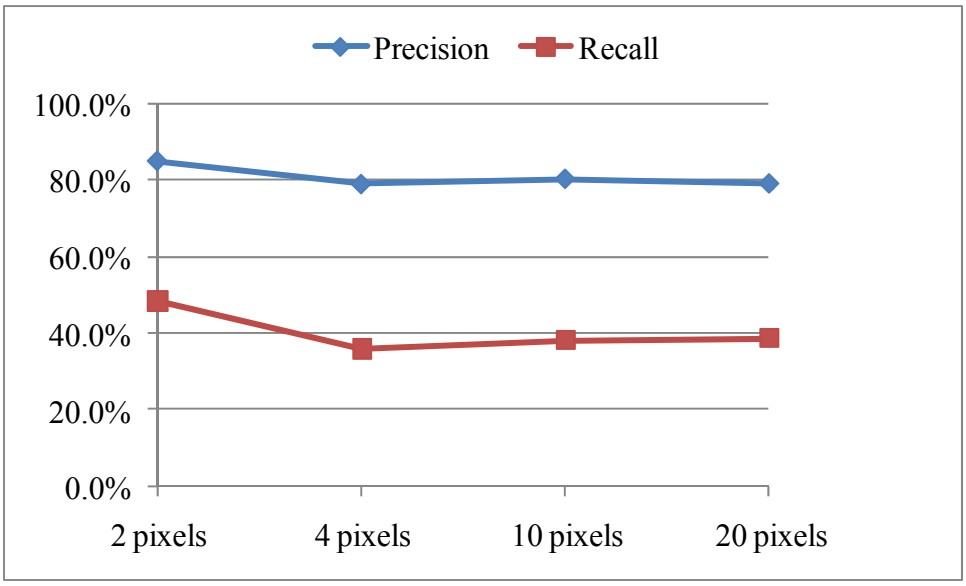


Figure 35: Precision and recall for column recognition under horizontal (0°) linear shifts

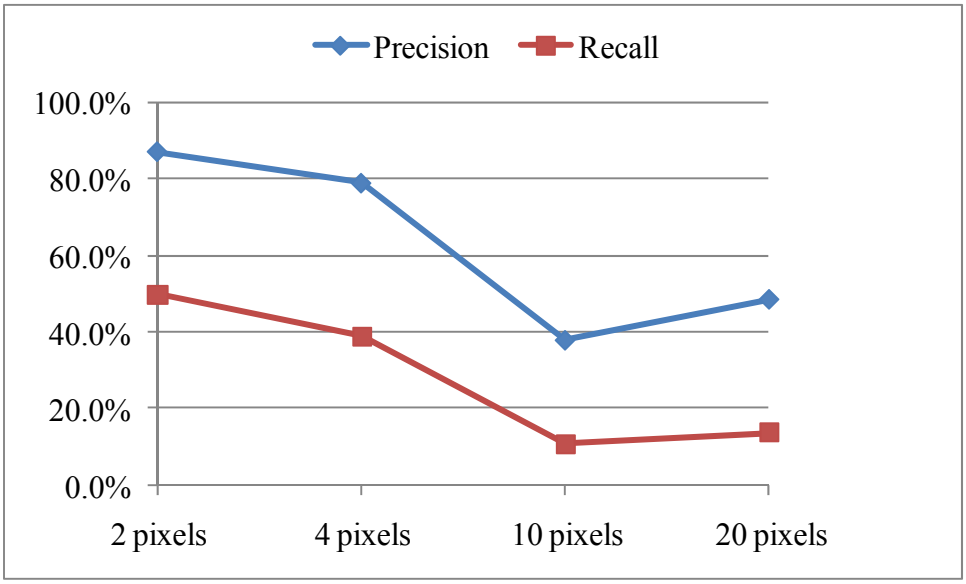


Figure 36: Precision and recall for column recognition under diagonal (45°) linear shifts

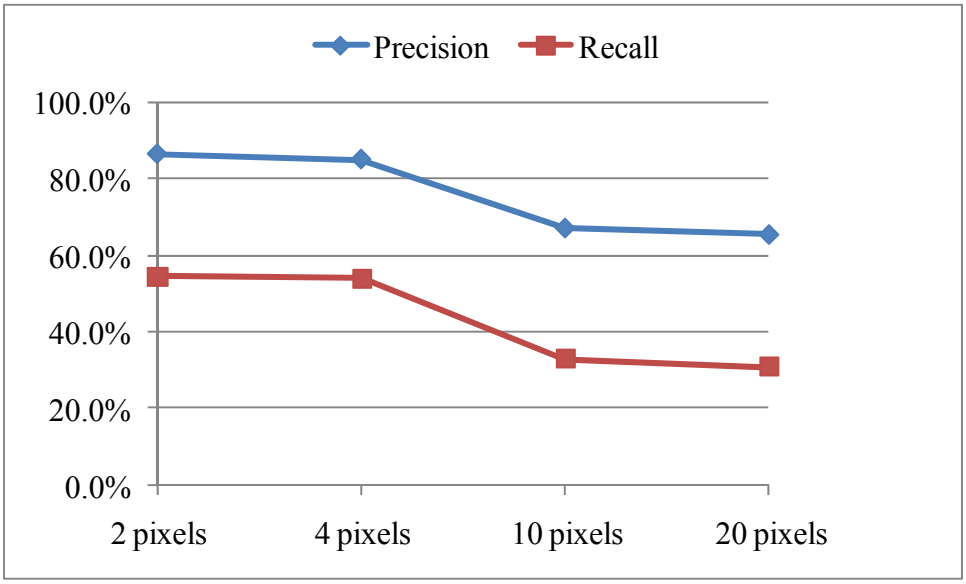


Figure 37: Precision and recall for column recognition under vertical (90°) linear shifts

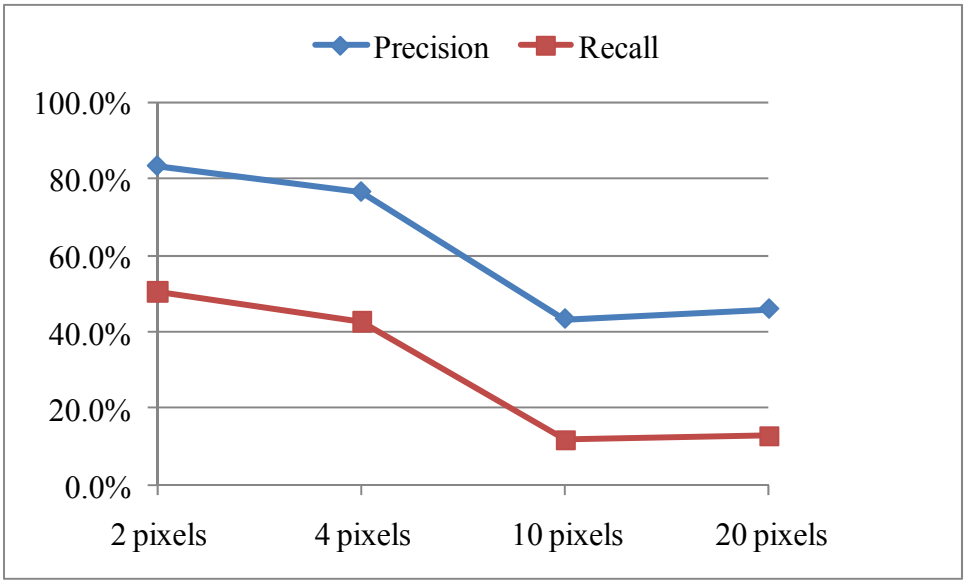


Figure 38: Precision and recall for column recognition under diagonal (135°) linear shifts

Aside from the motion blur, blur may be introduced into images or videos due to an out-of-focus shot with misplaced focal points. If the focal point of a shot is misplaced on the background instead of the concrete columns in an image or video frame, the background becomes clear and the concrete columns are blurry. Technically, this type of blur can be simulated by convolving an image or video frame with the filter that depends on the distance of each image point and corresponds to the image or video frame of an out-of-focus point source taken with a real camera (Potmesil and Chakravarty, 1982). In this research study, the blur is approximated by convolving the test images and video frames with circular averaging filters consisting of different radius size (i.e. uniform disk filters). Figure 39 illustrates one example of a concrete column blurred with a uniform disk filter.

(a) Original image

(b) Blurry image

Figure 39: Concrete column with blur

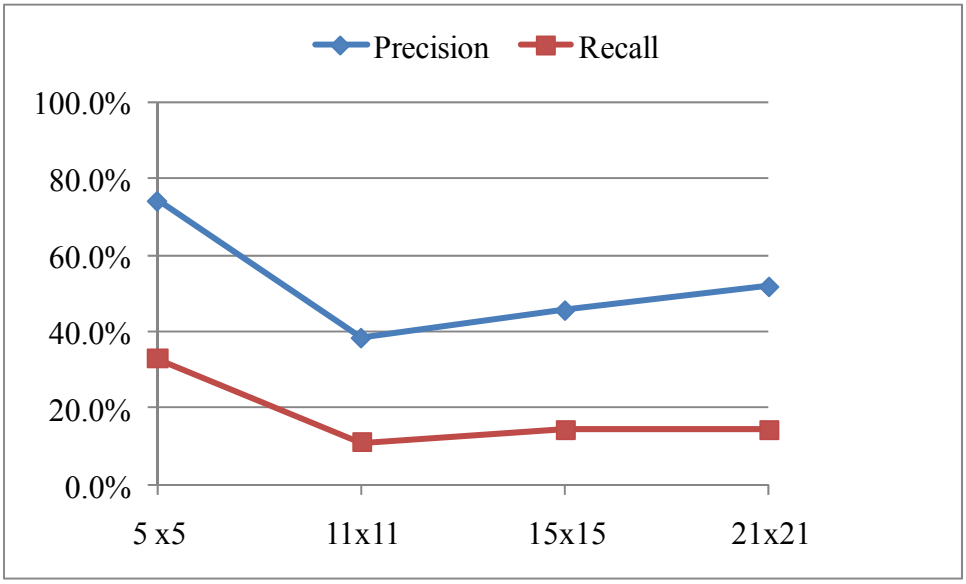


Figure 40: Precision and recall for column recognition under disk uniform filters with different size

The average precision and recall for recognizing concrete columns in the blurry images and videos are illustrated in Figure 40. According to the test results, it can be seen that the number of the real concrete columns that are correctly recognized by the approach is reduced with the increase of the filter size. This is because the larger the size of the filter, the more blurry the images or video frames are produced, and therefore the

more difficult it is to correctly recognize the real concrete columns in the images or videos. As for recognition precision, the precision is first reduced, when the size of the uniform disk filter is increased from 5x5 to 10x10. Then, the precision gradually increases. The reason for the increase is because the number of the incorrectly recognized concrete columns is reduced, while the number of the correctly recognized concrete columns is still maintained during the test.

Occlusion refers to the state of a concrete column partially or fully blocked. Consider the fact that the approach relies on two long near vertical lines as one of the necessary recognition conditions. Any vertical blocks that break this condition are not considered here. Instead, the focus of the occlusion test has been placed on horizontal blocks. The blocks are artificially introduced by drawing black boxes horizontally in the test images or video frames. There are five occlusion levels, which cover 10%, 20%, 30%, 40%, and 50% of the images or video frames horizontally (Figure 41).

(a) 10% occlusion (b) 20% occlusion (c) 30% occlusion

(d) 40% occlusion (e) 50% occlusion

Figure 41: Concrete columns with occlusion at five levels

The average precision and recall for recognizing concrete columns in the images and videos with different levels of horizontal blocks are illustrated in Figure 42. Both precision and recall drop with the introduction of occlusion. The more concrete column areas are occluded, the lower the precision and recall reach. When more than 50% areas of images or video frames are blocked, no concrete columns can be recognized by the method either correctly or incorrectly. This makes both precision and recall reach 0%.

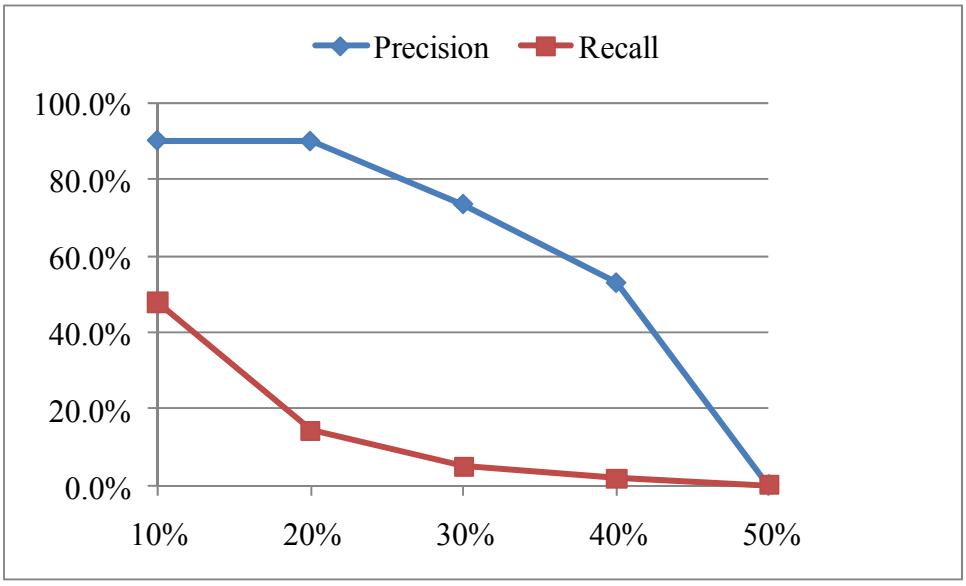


Figure 42: Precision and recall for column recognition under occlusion

Zooming is necessary in order to obtain the detailed defects/damage information inflicted on concrete column surfaces. When the lens of a camera zooms in, small parts of concrete columns are captured, but they are represented by large image or video frame regions. As a contrast, when the lens zooms out, a whole concrete column surface can be spotted, but they only cover small image or video frame regions (Figure 43).

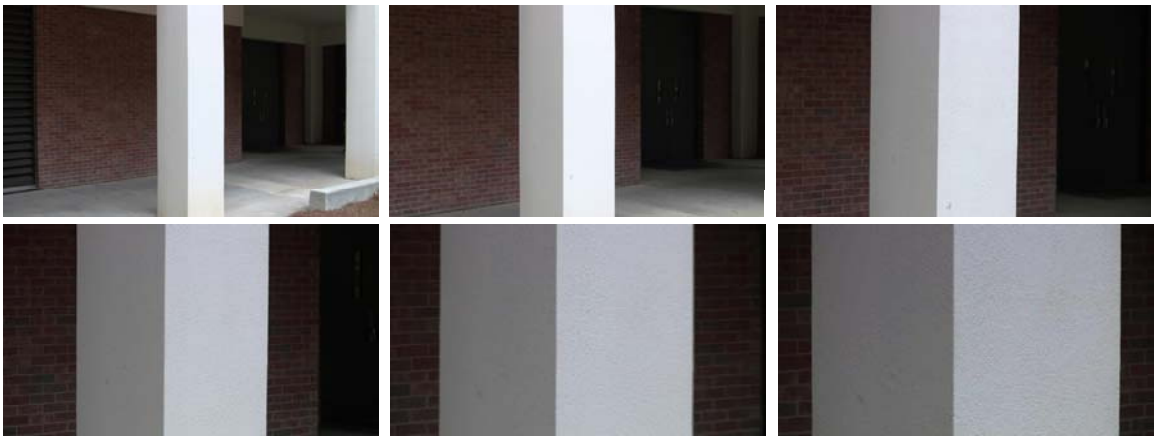


Figure 43: Concrete columns with zoom in and out

In order to test the performance of the approach in recognizing concrete columns when the lens of a camera zooms in or out, the recognition precision and recall have been recorded at different column detail levels. In this research study, the column detail levels are not completely determined by the lens' magnification factor. Other issues, such as the shooting distance of a camera and the dimensions of concrete columns, also affect how much detailed information can be shed from an image or video frame. Therefore, the column detail levels here are measured by the percentage of the areas covered by the concrete columns in an image or video frame. The larger (or smaller) the percentage, the higher (or lower) the column detail level.

The average precision and recall for recognizing concrete columns in the images and videos with different column detail levels are illustrated in Figure 42. According to the test results, it can be seen that the recognition precision is maintained around 90% and the recognition recall increases from 60% to 100%, when concrete column surfaces cover from 30% to 60% of the areas in an image or video frame. The reasons for the high recognition precision and the increase of the recall are because most images or video frames typically capture one real concrete column during the test, and moreover the

concrete column in each image or video frame is represented by a large concrete material region. This is helpful for the approach to correctly recognize the column.

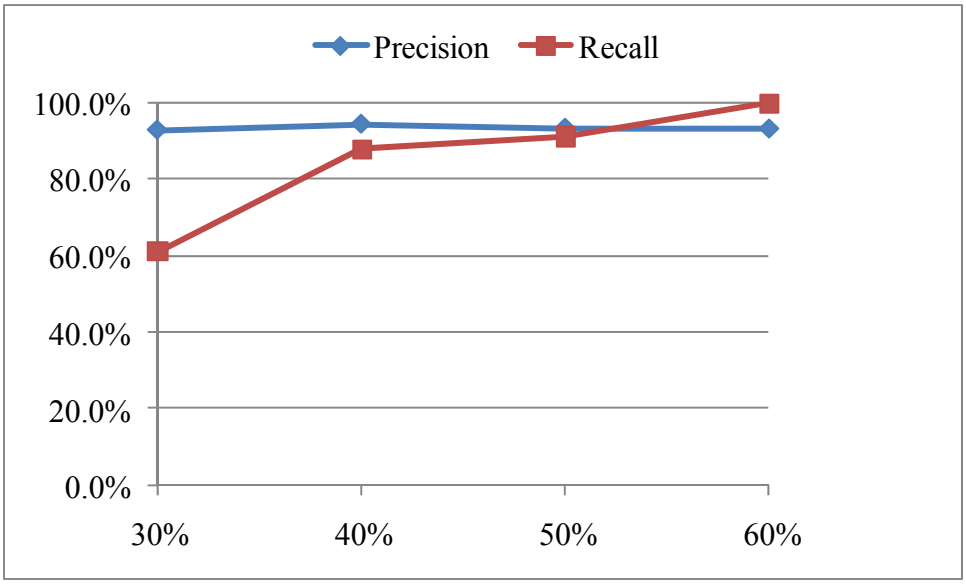


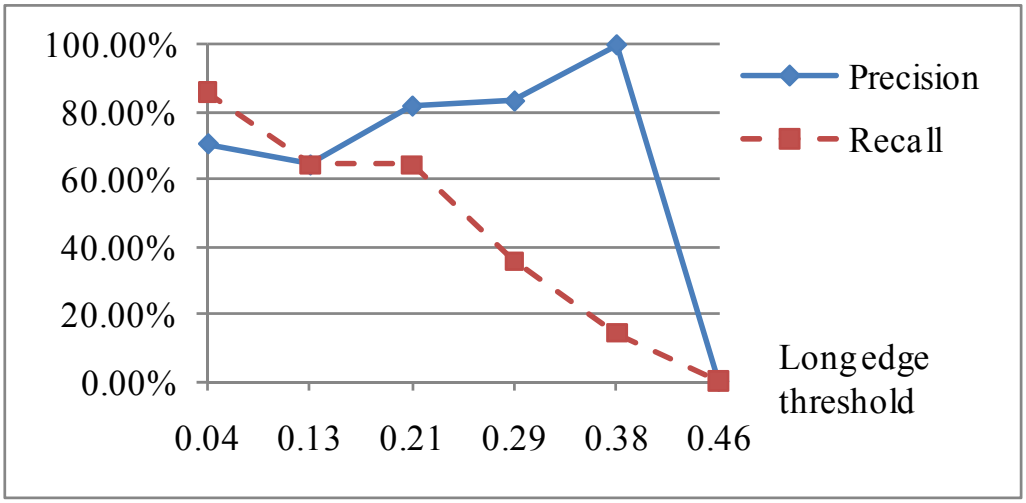
Figure 44: Precision and recall for column recognition under different detail levels

3.5 Discussion

In addition to the five metrics mentioned above, the hypothetical threshold for determining long near-vertical lines has a critical impact on the performance of recognizing concrete columns. In order to measure this impact, the hypothetical threshold is manually changed. In parallel, the number of correctly recognized column lines, the number of recognized column lines, and the number of real column lines are identified. The precision and recall of the recognized column lines are calculated as the ratio of the number of correctly recognized column lines over the number of recognized column lines and the ratio of the number of correctly recognized column lines over the number of real column lines. In Figure 45, it can be seen that when the hypothetical threshold is small, many real column lines can be found (relatively high recall) but many column lines are falsely recognized (relatively low precision) at the same time. When we increase the

hypothetical threshold, less real column lines are found. After reaching a certain threshold (0.46), no column lines can be found, which leads to zero in both precision and recall.

(a) Multiple concrete columns



(b) Precision and recall for column line recognition

Figure 45: Impact of the hypothetical threshold determining long near-vertical lines

Concrete walls whose width/length ratio is smaller than 1 are sometimes falsely recognized as concrete columns. This false recognition ratio can be reduced by adjusting the assumption of the aspect ratio (width/length) of the concrete columns. Considering it is common to see the concrete columns have the aspect ratio of 1:6, 1:7 or even 1:10 in

practical structures, the assumption of the aspect ratio (1:1) of concrete columns made in the method is conservative.

The method proposed in this research study only investigated the recognition of concrete columns in structures. The method can be extended to recognize the columns with other construction materials. For example, the method can be used to recognize steel columns by retraining the classifier to produce the decision boundaries of steel samples in the feature space. Figure 46 is an example of recognizing a steel column being erected at a construction site.

Figure 46: Example of recognizing a steel column

3.6 Summary

This chapter described the approach of recognizing concrete columns in images and videos. In the approach, the near-vertical edge pixels are first extracted, and the Hough Transform is used to construct long near-vertical lines. The material information contained in the region of two long near-vertical lines is retrieved. If the ANN classifier determines the material is concrete, a concrete column is therefore recognized.

The approach was implemented in a Visual Studio .NET environment. Real images and videos were used to test the effectiveness of the method. The test results were compared with manual recognition results to find the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) of the method. The recognition precision (TP / (TP+FP)) and recall (TP / (TP+FN)) were then calculated. According to the test results, the prevision and recall can reach 89.1% and 79.1%.

The approach was further measured from five aspects: illumination, jittering, blurring, occlusion, and zooming. The recognition precision and recall ratios in these five aspects were recorded. According to the results, it was found that the approach could not correctly recognize most real concrete columns in overly bright lighting (Level 5). Weak lighting (Level 1) also limited the approach in recognizing real concrete columns, but the recognition made by the approach in weak lighting was always correct. As for the linear motion of a camera, the approach was more sensitive to the diagonal and vertical camera shifts than the horizontal camera shift. The approach can recognize the concrete columns in the slightly blurry images or video frames. When the blurry level increased, the recognition precision increased but the recall decreased. Occlusion reduced the ability of the approach for concrete column recognition both in precision and recall. When the

occlusion area was more than 50% of an image or video frame, concrete columns could no longer be recognized. The approach could maintain high recognition precision and gradually increase recognition recall, when the camera was shooting at one concrete column and then zooming in for the detailed defect/damage information that lies on the column surfaces.

Although the approach proposed in this research study is limited to the recognition of concrete columns in structures, it is convenient to be expended to recognize columns with other materials such as steel or wood. One necessary step is to retrain the material classifier by providing appropriate material training samples.

CHAPTER 4

DEFECTS/DAMAGE DETECTION, PROPERTIES RETRIEVAL, AND ASSESSMENT

This chapter describes two novel methods in the area of defects and damage detection, properties retrieval, and impact evaluation. The first method is designed to retrieve the properties of the cracks on concrete column surfaces. The method starts with producing the crack map of a concrete column surface using state-of-the-art crack detection techniques. Then, the topological skeletons of the cracks in the map are extracted through binary image thinning. The distance of the crack pixels to the crack boundaries is calculated using a distance transform. According to the skeleton configurations and the distance value of each crack pixel, the properties of the cracks in the map, such as maximum width, length, and orientation, can be measured.

The second method is focused on the detection, properties retrieval, and evaluation of air pockets and discoloration that are inflicted on concrete surfaces. A spot filter is first designed to locate the air pockets on the concrete surfaces. Meanwhile, the discoloration areas on the concrete surface are identified using image segmentation techniques. The properties of the air pockets (e.g. number and size) and discoloration (e.g. covering area) are then calculated. These properties can quantify the visual impact of the air pockets and discoloration to evaluate the visual quality of the concrete surfaces in terms of air pockets and discoloration.

Both methods were implemented using Visual Stuido .NET. The methods were tested using real concrete surface images with cracks, air pockets, and/or discoloration. The

results from the methods were compared with the results from manual cracks, air pockets, and/or discoloration detection and evaluation to validate the effectiveness of the methods.

4.1 Crack Detection and Properties Retrieval

Many crack detection methods have been developed to produce the crack maps on the surfaces of structural elements. The percolation based method proposed by Yamaguchi and Hashimoto (2009) is adopted here due to its robustness to image noise and fast crack detection speed even on a large-scale concrete surface. The idea behind the percolation based crack detection method is from the natural phenomenon of liquid permeation through a crack on a concrete surface. Imagine water is poured at crack boundaries, and it will always make its way to fill the cracks. As a contrast, if water is poured on a concrete surface, it will be spread evenly as a circle. The percolation in this research study is performed only at the image pixels that have high gradient magnitudes, in order to reduce the computation load of initiating a percolation process at each pixel of a concrete surface image. This is because the crack boundaries in the concrete surface image are always characterized by large first derivatives which result in high magnitudes along certain particular directions.

When a crack map is produced, it is necessary to isolate each crack from the map, and retrieve its properties that are useful for evaluating the damage state of the structural elements in terms of cracks. The properties investigated in this research study include crack length, orientation, and maximum width. Consider the properties are measured at the level of image pixels. The retrieved properties are spatially correlated with the dimension and orientation of the structural element surfaces that the cracks lie on. This

way, the relative measurements of the crack properties are produced. The damage state of the structural element can be estimated using the relative measurements in practice.

In order to retrieve the properties of the cracks in the crack map, a binary image thinning algorithm (Momma, 2008) is first applied in the map to retrieve the crack skeleton points. Also, a Euclidean distance transform (Bradski and Kaehler, 2008) is used to calculate the distance field, which supplies each crack pixel in the map with the nearest distance to the crack boundaries. The crack skeleton points and the distance values from the skeleton points to the crack boundaries fully support the representation of a crack since they contain all the information necessary to reconstruct the crack (Figure 47).

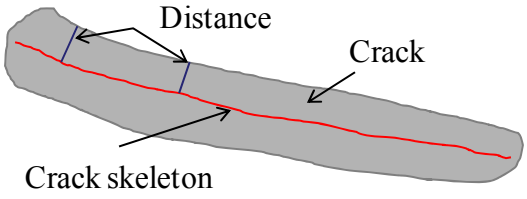


Figure 47: Crack reconstruction with skeleton and distance information

The topological configuration of a crack is ascertained by checking the connectivity of its crack skeleton points in the map. This process is illustrated in Figure 48. Suppose one crack skeleton point is visited, and its neighboring skeleton points are checked. If there is only one crack skeleton point connected to the point, the current crack segment grows by including that neighboring skeleton point. If there are two or more crack skeleton points connecting to the point, the current crack segment stops growing, and new segments are then created. The number of the newly created segments depends on the number of the neighboring crack skeleton points. For example, two segments (S2 and S3) are created at the crack skeleton point 5 in Figure 48, since there are two skeleton points (6 and 7) connecting to point 5. The new segments start to grow by visiting the remaining skeleton

points. When all the skeleton points are visited, the direction of all the segments is checked. Any two segments are merged if they have the same direction, and at the same time the end point of one segment is specified as the start point of the other.

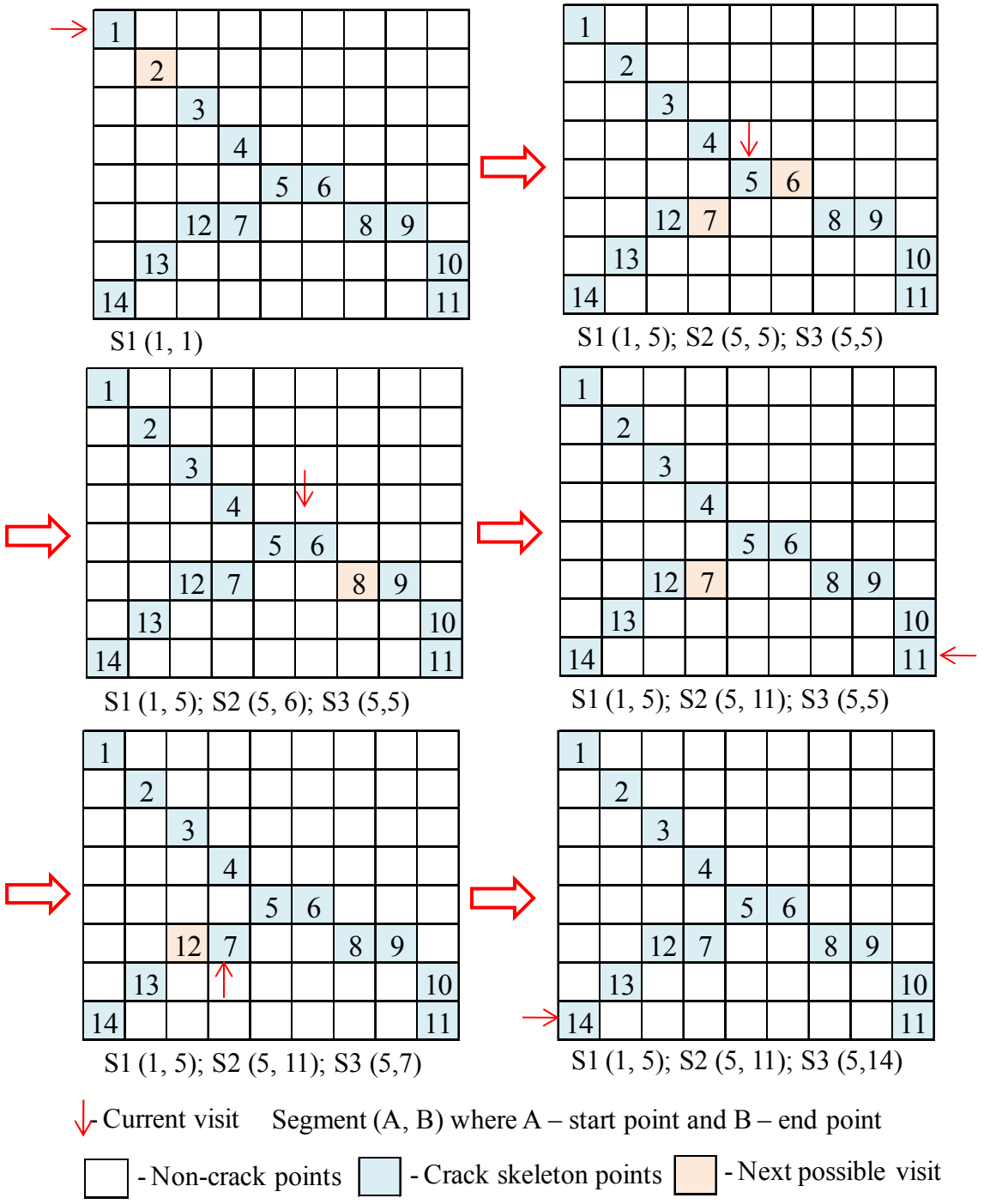


Figure 48: Crack topological configuration

The properties of a crack are retrieved based on its skeleton configuration information and distance field. The crack length is equivalent to the length of the crack skeleton, which is approximated by the height of an object-oriented bounding box that circumscribes the crack skeleton points. The crack orientation is indicated by the direction of the object-oriented bounding box. The double of the largest distance that exists at skeleton points represents the maximum width of the crack.

All values are measured at the level of image pixels and are of little value to estimate the damage state of real structural members until they are spatially correlated with the dimension and orientation of the structural members. Spatially correlating the cracks to the structural members produces relative measurements. Take concrete columns for an example. The following measurements are calculated including: 1) the angle of crack's direction in relevance to the column's vertical lines, 2) the projection of the crack's length on the column's width, and 3) the largest crack's width in relevance to the column's width. The relative measurements alleviate the issues of the retrieved properties being controlled by camera settings and image distortions (Figure 49).

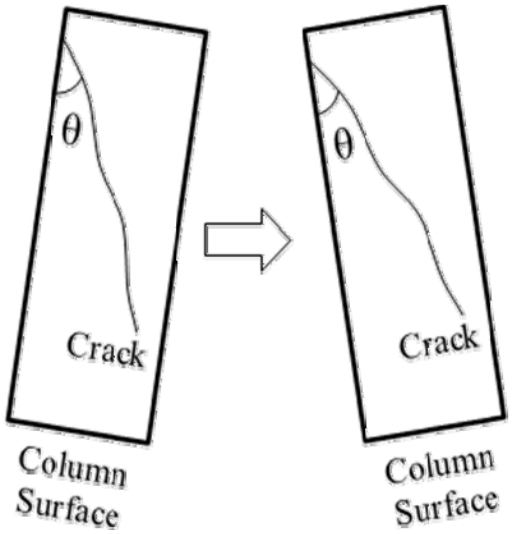


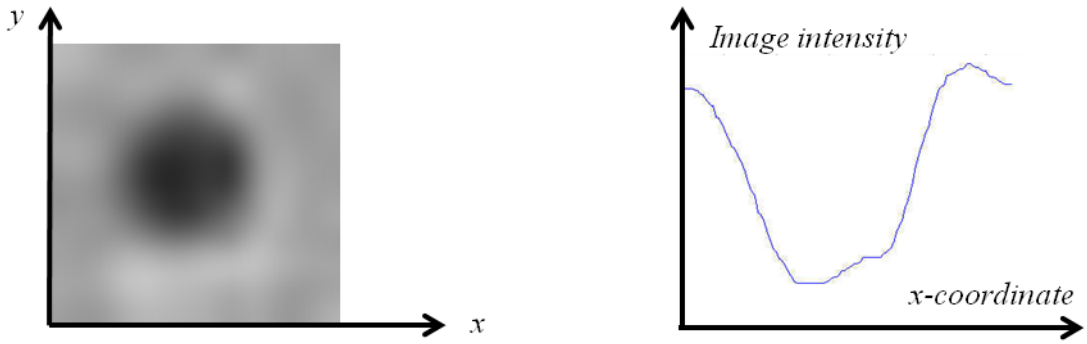
Figure 49: Invariant crack orientation under different camera orientations.

4.2 Air Pockets Detection and Properties Retrieval

Air pockets are one kind of common concrete surface defects. Any mistakes during construction, such as inadequate concrete consolidation and improper concrete curing will produce air pockets in a concrete product. Their existence undermines the desired appearance and visual uniformity of architectural concrete. Therefore, they are always in the checklist of a concrete construction inspector. In this research study, a novel method of air pockets detection and properties retrieval is proposed. The method includes spot filtering, image scaling, and visual impact approximation.

4.2.1 Spot Filter Design

To locate air pockets in a concrete surface image, the unique characteristic of the air pockets are considered first. Typically, the shape of an air pocket on the concrete surface looks like a spot with the image inverse intensity values. Specifically, the intensity values of the air pocket in the concrete surface image is changing from the dark at the center of the air pocket to the bright at the air pocket's perimeter, until it has the same intensity as the concrete surface as shown in Figure 50.

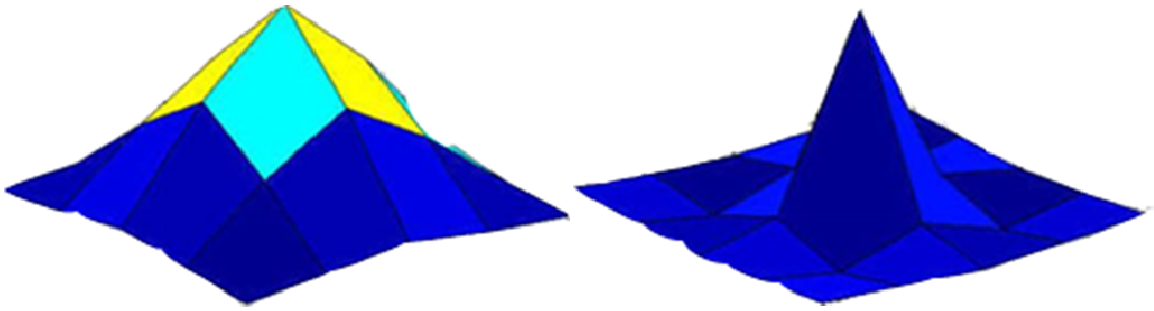


(a) an air pocket in a gray-scale image

(b) the change of intensity values along the x direction

Figure 50: Visual characteristics of air pockets

According to this visual characteristic, two filters are designed to measure the spottiness of an image. They are used as the candidate filters to locate air pockets. The first spot filter is formed with a weighted sum of three concentric, symmetric Gaussian filters with weights 1, -2, and 1. Their corresponding sigmas are 0.62, 1 and 1.6. The second spot filter is given by a weighted sum of two concentric symmetric Gaussians, with weights 1 and -1. Their corresponding sigmas are 0.71 and 1.14. The 3D representations of the two filters are illustrated in Figure 51.

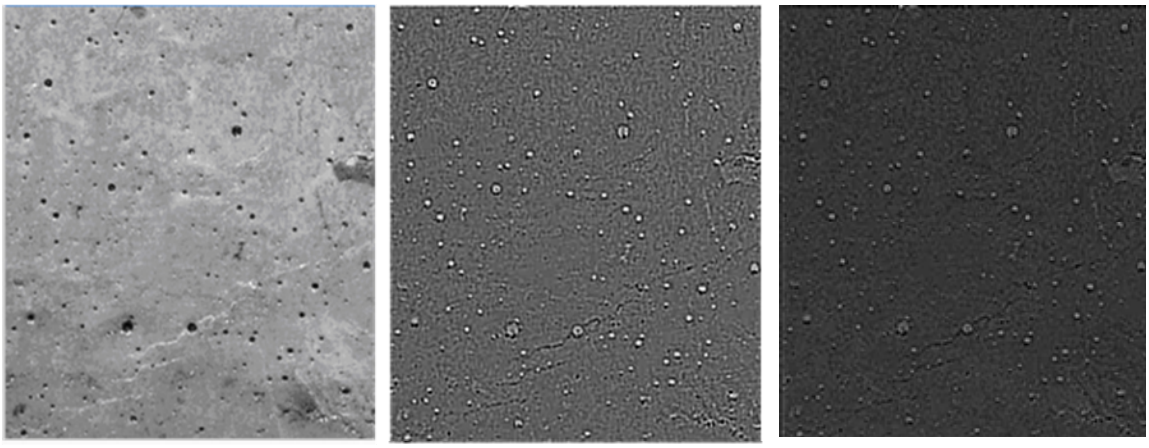


(a) the filter composed of three Gaussian filters

(b) the filter composed of two Gaussian filters

Figure 51: Spot filters

When both spot filters are applied in the images composed of different kinds of air pockets, the response values of the images to these filters are retrieved. One example is shown in Figure 52, where Figure 52 (a) is the original image, and the response values of the image to the spot filters are illustrated separately in Figure 52 (b) and 52 (c). The white regions in Figure 52 (b) and 52 (c) indicate that the image has strong responses to the filters. It can be seen that the locations where the high response values are produced "happen" to be where the air pockets exist. This way, the air pockets on a concrete surface image can be detected by finding its high response values after the spot filtering.



(a) Original concrete surface image

(b) Response from the filter in Figure 51 (a)

(c) Response from the filter in Figure 51 (b)

Figure 52: Spot filtering with two designed filters

4.2.2 Image Scaling

If one spot filter is applied directly in a concrete surface image, only the air pockets that have a similar size as the filter can be accurately detected in the image. However, other air pockets cannot be detected and represented accurately. This point is clearly illustrated in Figure 53. A small real air pocket may be represented by a large air pocket in the detection results, while a large real air pocket may be represented by two, three or even four small air pockets. This affects the detection accuracy of the filter, since many faked air pockets are introduced. Also, it leads to the error of counting the number of the air pockets detected by the filter and the mistake of approximating the size of the air pockets. This limitation can be overcome with the introduction of the concept of an image pyramid.

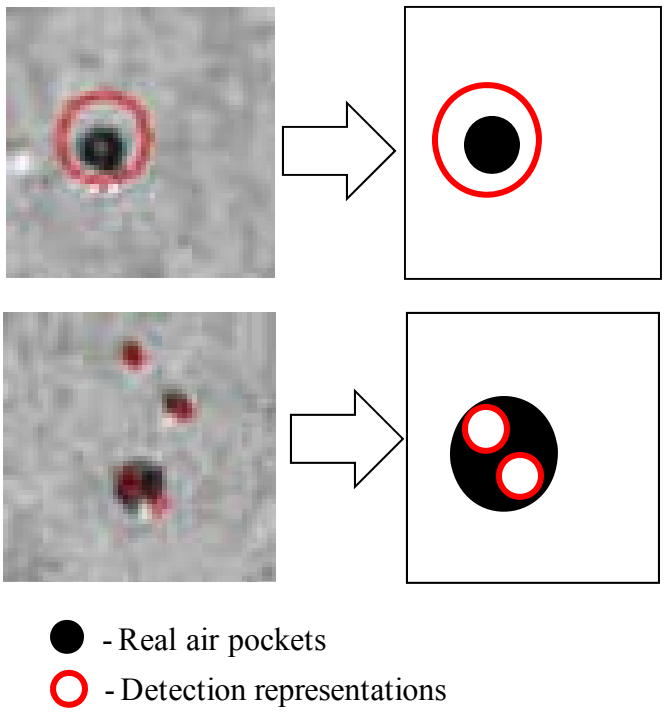


Figure 53: Error of detecting air pockets

The pyramid is an artificial image hierarchy. One example of the pyramid is shown in Figure 54. At each level of the pyramid, an original image is scaled down to a certain percentage in size. When the size of the original image is scaled down, the size of the air pockets in the image is also reduced. This makes large air pockets in the low level of the pyramid become small in the high level. The benefit is that the air pockets whose size is similar to the size of the filter in the low level of the pyramid cannot be detected by the filter any more, if the size of the image is reduced enough. As for the air pockets whose size is larger than the size of the filter before, they can have the similar size as the filter and therefore be successfully detected, if an appropriate scaling percentage is selected. This way, both small and large air pockets can be detected by one filter, when the filter is applied at each level of the pyramid (Figure 55).

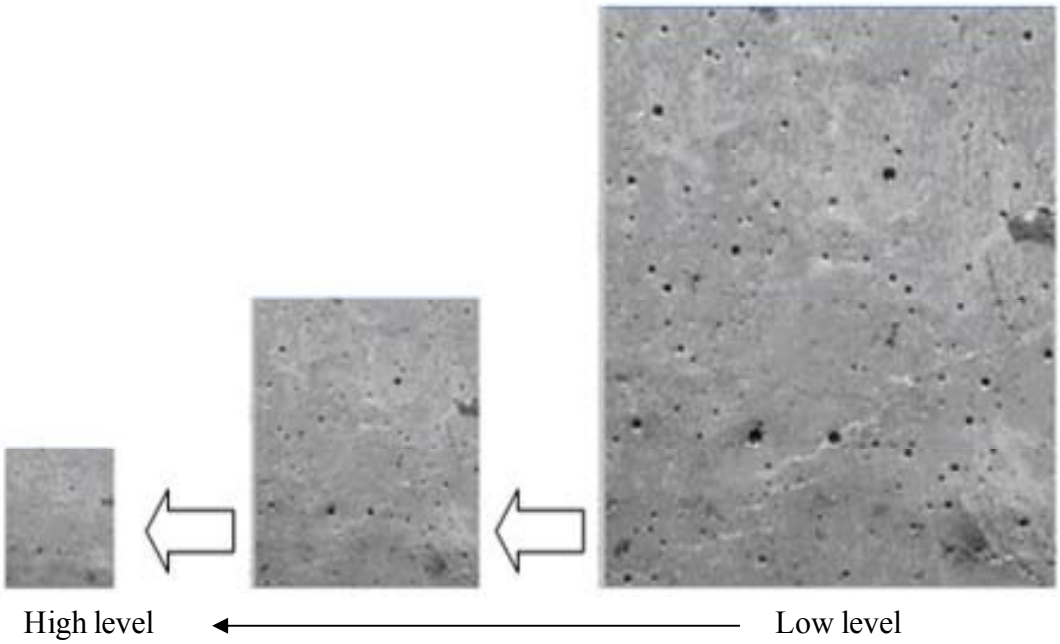


Figure 54: Image scaling with three levels

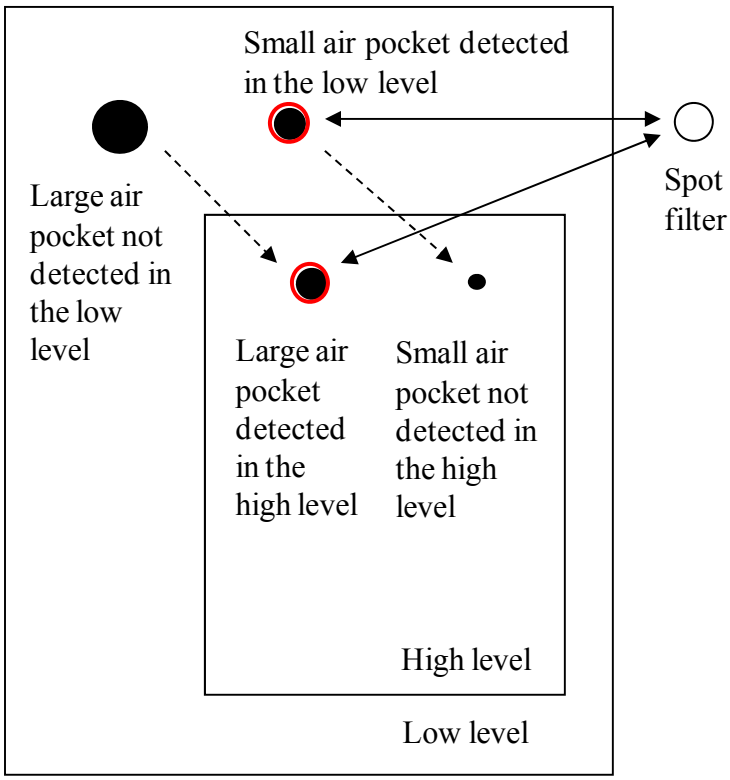


Figure 55: Detecting air pockets with different size

4.2.3 Visual Impact Ratios of Air Pockets

When applying the filter into the pyramid of a concrete surface image, high responses are expected at each level of the pyramid. Therefore, the properties of the air pockets in the concrete surface image can be retrieved as follows. First, the position of the air pockets can be automatically located by finding the position of the high responses. The number of the air pockets can be calculated by automatically counting the number of the high responses. Also, the size of the air pockets detected at one level of the pyramid can be approximated through dividing the size of the filter by the scaling percentage adopted at the level. For example, suppose one air pocket is detected by the filter, the size of which is 5 pixels, at the pyramid level with scaling percentage 0.5. Then, the size of the air pocket is determined as 10 (5/0.5) pixels. When the size of an air pocket is known, the covering area of the air pocket can be calculated as the area of a circle. The diameter of the circle is the size of the air pocket. The total covering area of the air pockets in the concrete surface image is measured by adding the area of each individual air pocket.

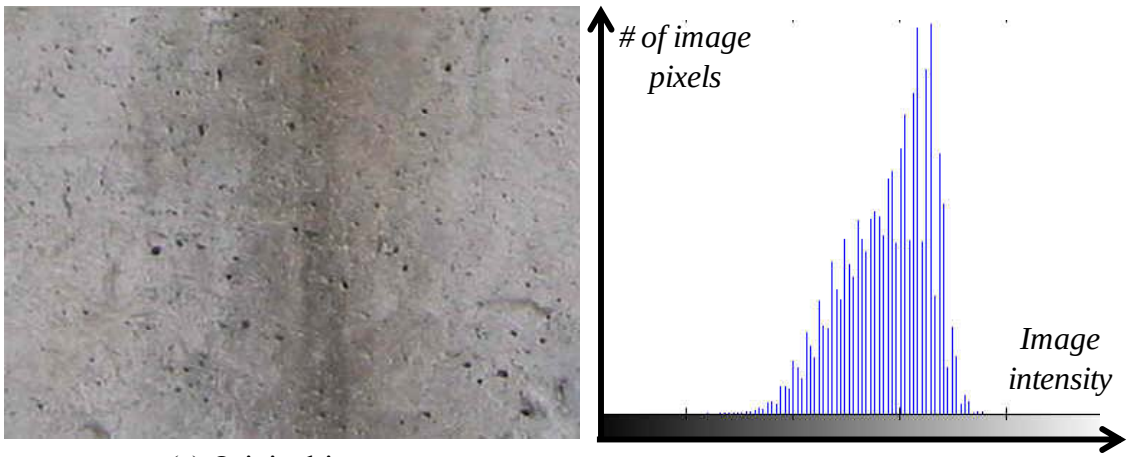
In order to assess the visual quality of a concrete surface, two visual impact ratios of the air pockets (VIRAP1 and VIRAP2) are further calculated. The VIRAP1 is the percentage of the concrete surface that is covered by the air pockets over the total area of the concrete surface. The VIRAP2 is calculated by dividing the VIRAP1 by the number of air pockets on the surface.

Both ratios indicate the impact of the air pockets on the visual quality of a concrete surface. When the VIRAP1 is large, it means that the air pockets cover a large concrete surface area. When the VIRAP1 is small, the air pockets cover a small concrete surface area. Given a certain VIRAP1 (i.e. a fixed concrete surface area covered by the air

pockets), the large VIRAP2 means that the number of the air pockets covering the area is small. Therefore, the air pockets may be composed of large air pockets, which have a significant visibility issue. The small VIRAP2 means that the large number of the air pockets covering the area. Therefore, the air pockets may be composed of small air pockets, and the visibility issue is not significant. In one word, the smaller both ratios are, the higher the visual quality of the concrete surface.

4.3 Discoloration Detection and Properties Retrieval

Discoloration is the departure of color from that the normal or desired concrete surface (ACI 116R-00, 2005). One image example of a concrete surface containing discoloration is shown in Figure 56. It can be seen that the discoloration at the concrete surface has no specified shape, no sharp boundaries, and no certain color or texture. Moreover, according to its gray-scale histogram, no obvious threshold can be selected to differentiate the discoloration areas from the normal concrete area.



(a) Original image (b) Histogram of image intensity

Figure 56: A concrete surface images containing discoloration

Considering all the difficulties, the discoloration is detected with two steps. First, the degree of the discoloration on the concrete surface is globally measured by calculating the standard deviation of the color values of the concrete surface image. The larger (smaller) the standard deviation is, the less (more) the color uniformity of the inspected concrete surface. When the standard deviation is large enough, image segmentation and classification techniques are used to differentiate the discoloration areas from the concrete surface.

Here, the image segmentation algorithm developed by Felzenszwalb and Huttenlocher (2004) is used. In the algorithm, the graph-based representation of an image is first constructed, where each pixel in the image is regarded as a node in the graph and its neighboring pixels are assumed to be connected to the pixel with edges. The weight of an edge is determined by the color difference between two connecting pixels. The more (less) the color difference between the two pixels, the higher (lower) the weight is. Whether the pixel is merged with its neighboring pixel or split from it is based on the weight of their edge. Figure 57 illustrates one example of image segmentation results, where an image is divided into multiple regions.

(a) Original image (b) Image segmentation

Note: different colors are used to represent different regions

Figure 57: Concrete surface segmentation

In order to identify the discoloration regions in the image, the image region composed of the largest number of image pixels is selected as the seed region first. Other regions, regardless of their connectivity to the seed region or not, are compared with the seed region to find whether they have the similar color characteristics (red, green and blue) as the seed region. If they have, the regions are classified into one class called the seed region class. Or else, they are classified into the non-seed region class (Figure 58).

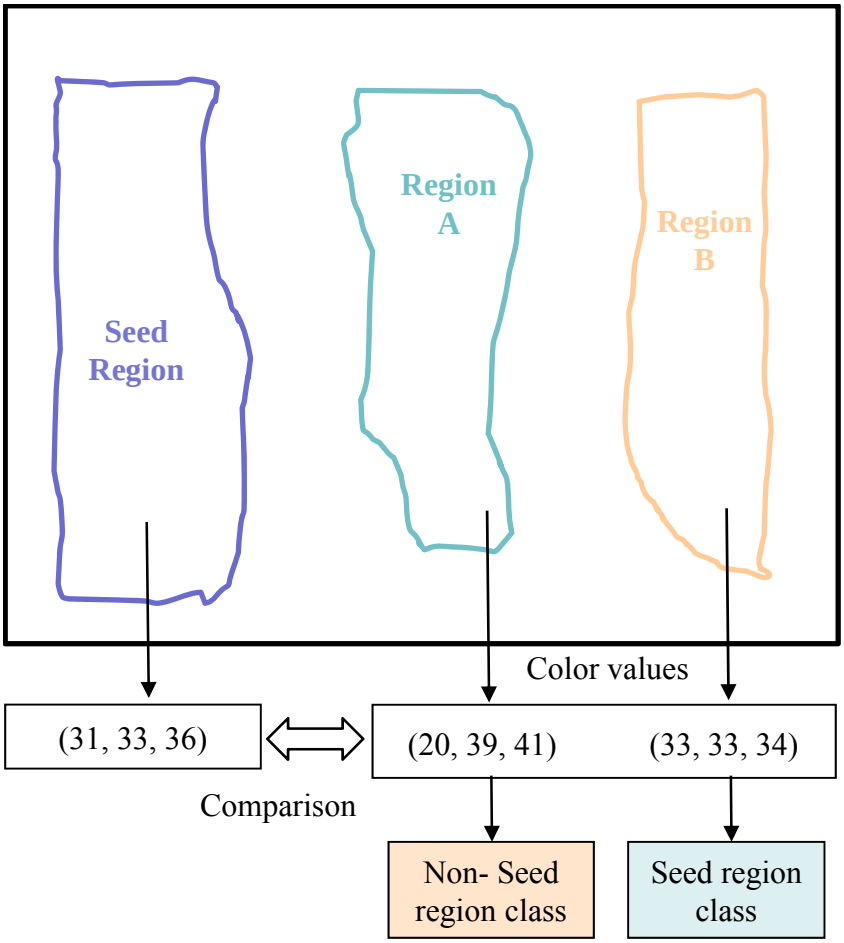


Figure 58: Region classification

After all the regions are classified into the seed region class or non-seed region class, the number of the image pixels belonging to each class is counted. If the number of the image pixels belonging to the seed region class is larger than the number of the image

pixels belonging to the non-seed region class, the regions in the non-seed region class are identified as discoloration based on the assumption that *the discoloration area at a concrete surface is smaller than the non discoloration area*. The assumption is made because the color of concrete is rarely specified in a construction project. This leads to no "right" color for concrete, and the discoloration is simply the existence of two or more color tones at one concrete surface. As a result, the assumption that the surface area with the less dominant color is discolored is reasonable in this research study. Alternatively, a user can specify a desired color tone, which greatly simplifies the problem.

The discoloration properties investigated in this research study include the percentage of the discoloration areas on a concrete surface and the color departure degree of the discoloration areas from the normal concrete surface areas. When the discoloration regions are identified in a concrete surface image, the size of the discoloration regions can be approximated by the number of the image pixels in the discoloration regions. The percentage of the discoloration areas on the concrete surface can be calculated as the ratio of the number of the image pixels in the discoloration regions over the number of the image pixels of the whole concrete surface. The color departure degree of the discoloration areas can be measured as the color difference between the discoloration and non-discoloration regions using Eq. 4,

$$Diff = \sqrt{\left(\frac{\sum R_d}{N_d} - \frac{\sum R_{nd}}{N_{nd}}\right)^2 + \left(\frac{\sum G_d}{N_d} - \frac{\sum G_{nd}}{N_{nd}}\right)^2 + \left(\frac{\sum B_d}{N_d} - \frac{\sum B_{nd}}{N_{nd}}\right)^2} \qquad \text{(Eq. 4)}$$

Where R_d, G_d and B_d are the red, green and blue color of an image pixel in the discoloration regions; R_{nd}, G_{nd} and B_{nd} are the red, green and blue color of an image pixel

in the non-discoloration regions; N_d is the number of the image pixel in the discoloration regions; and N_{nd} is the number of the image pixel in the non-discoloration regions.

The color departure degree of the discoloration areas and the percentage of the discoloration areas are regarded as two visual impact ratios (VIRD1 and VIRD2) for the purpose of assessing the visual quality of concrete surfaces in terms of the discoloration. When the VIRD1 is large, it means that the discoloration departs from the normal concrete surface in the color characteristics significantly, and therefore the visual quality of the concrete surface in terms of the discoloration is poor in this case. When the VIRD2 is large, it means that the discoloration areas cover a large part of a concrete surface and the visual quality of the concrete surface in terms of discoloration is also poor. Only when both VIRD1 and VIRD2 are small, is the visual quality of the concrete surface in terms of the discoloration acceptable.

4.4 Visual Quality Assessment in terms of Air Pockets and Discoloration

The visual impact ratios of air pockets and discoloration (VIRAP1, VIRAP2, VIRD1, and VIRD2) are quantitative data. However, the condition of a concrete surface is rated qualitatively (good, satisfactory or poor), when making a condition survey of concrete in service (ACI 201.1R-92, 2005). In order to qualitatively evaluate the visual quality of concrete surfaces in terms of air pockets and discoloration, it is necessary to establish the link between the quantitative data and the qualitative measurements. In this research study, a survey is conducted, and experienced inspectors are involved into the process of establishing this link (Figure 59). The inspectors are provided with multiple concrete surface images, and they are asked to assess the visual quality of the concrete surfaces in the images manually in terms of air pockets and discoloration.

Survey for Concrete Surface Defects - Impact Assessment on the Appearance of Concrete Surfaces

The following concrete surface image samples have been shot from the same distance. Please rank them with respect to perceived quality in terms of air pockets and discoloration (1 being the best and 5 being the worst).

Also, please circle either yes or no depending on whether the sample would comply with the appearance specifications in a project.

Please send your answers back to zhzhu@umich.edu. Thanks.

SAMPLE 1

1 2 3 4 5 (air pockets)

1 2 3 4 5 (discoloration)

YES NO

Figure 59: A concrete surface survey

In the survey, the visual quality of a concrete surface is rated from 1 (being the best) to 5 (being the worst). The inspectors select one rating point according to their perceived surface visual quality in terms of air pockets and discoloration separately. They also decide whether the surface quality in an image is acceptable or not, depending on whether it complies with the concrete surface appearance specifications. Based on the manual assessment from the inspectors, an appropriate threshold for each visual impact

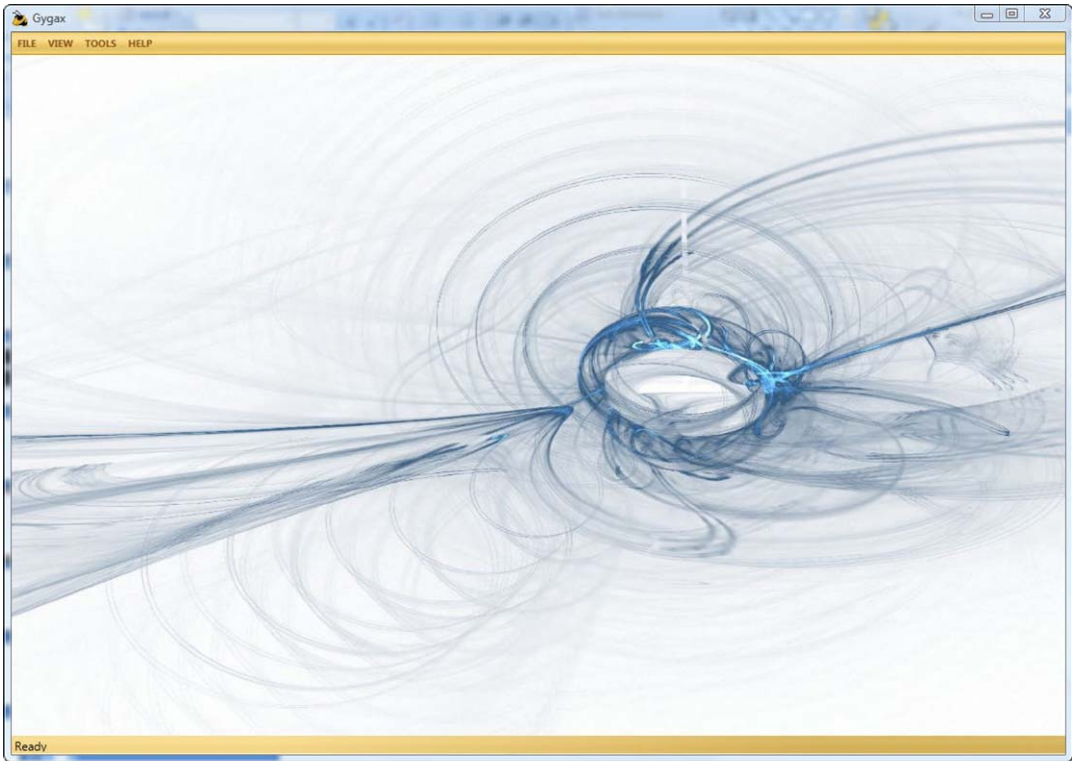
ratio (VIRAP1, VIRAP2, VIRD1, or VIRD2) is selected, so that these quantitative visual impact ratios can be used to assess the visual quality of the concrete surface qualitatively.

4.5 Implementation and Results

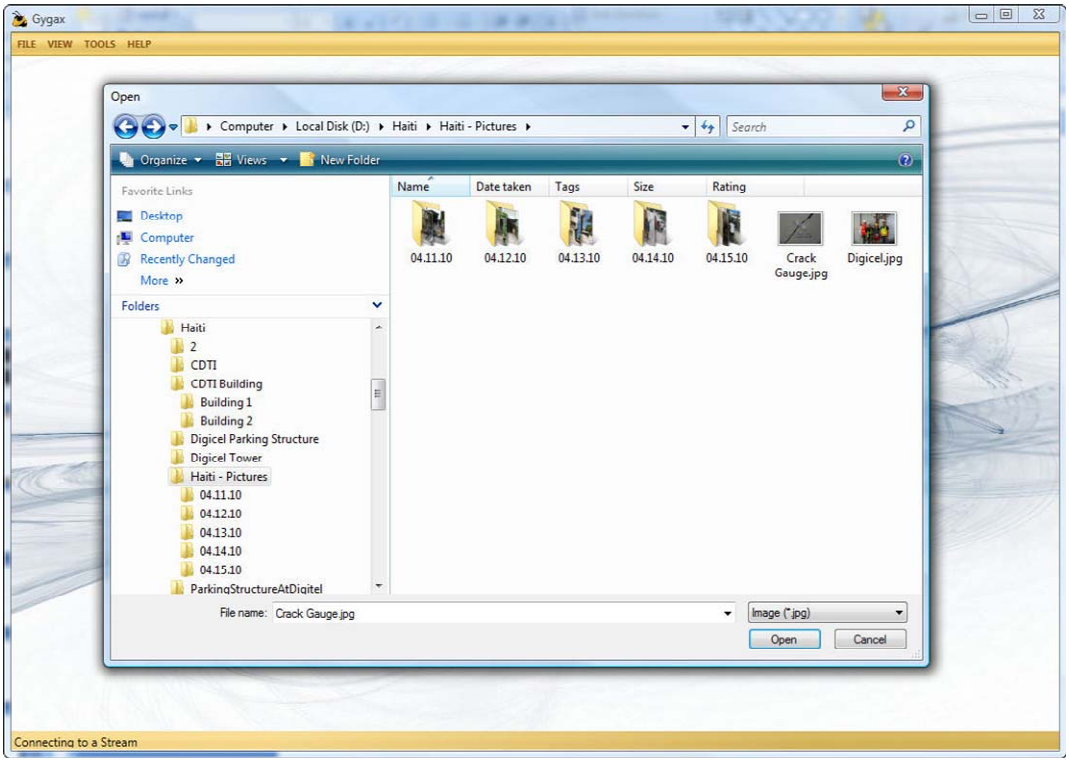
4.5.1 Implementation on Crack Properties Retrieval

A prototype was developed to retrieve the properties of the cracks inflicted on concrete column surfaces using Microsoft Visual .NET, Open CV, and EmguCV. Open CV is a collection of C functions and C++ classes. Many popular algorithms about image processing and computer vision have already been implemented there. Therefore, it is used as a main image processing library. EmguCV was used as a wrapper to allow OpenCV functions to be called in the Microsoft Visual .NET environment.

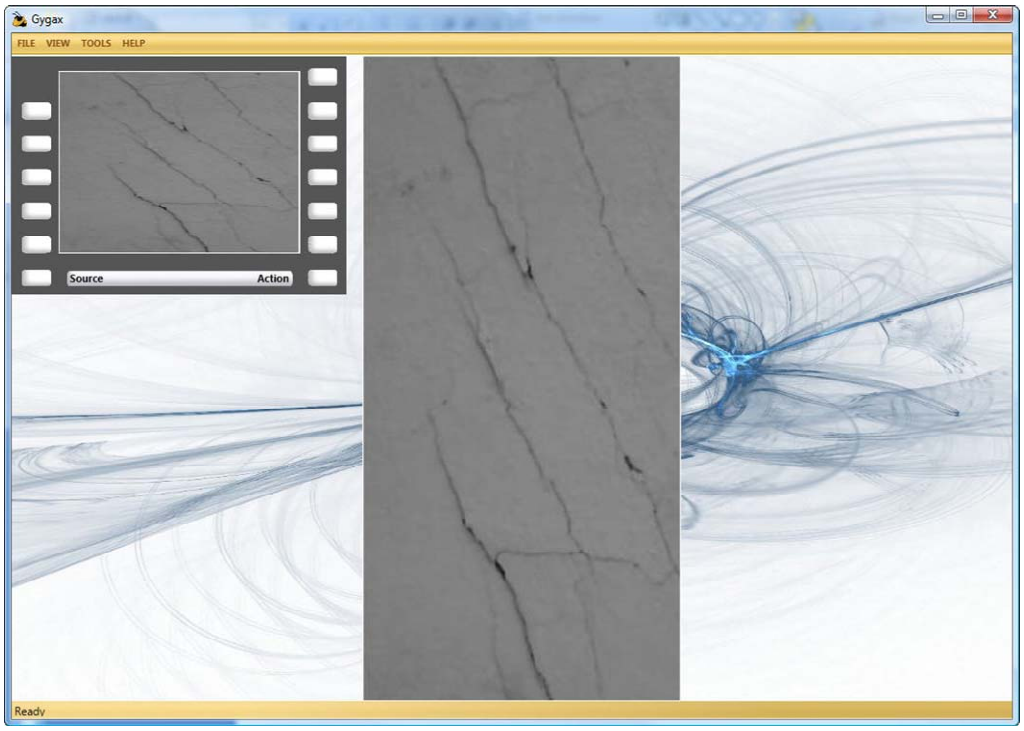
Figure 47 illustrates the screenshots of using the developed prototype to retrieve the crack properties for a concrete column surface image. Figure 60 (a) is the main interface of the prototype. A user can then browse an image folder to load one image into the prototype (Figure 60 (b) and (c)). When the user selects the "CRACK PROPERTIES RETRIEVAL" option from the menu, the corresponding crack detection result (i.e. a crack map) is displayed, while the corresponding crack properties are saved in a separate text file (Figure 60 (d)).



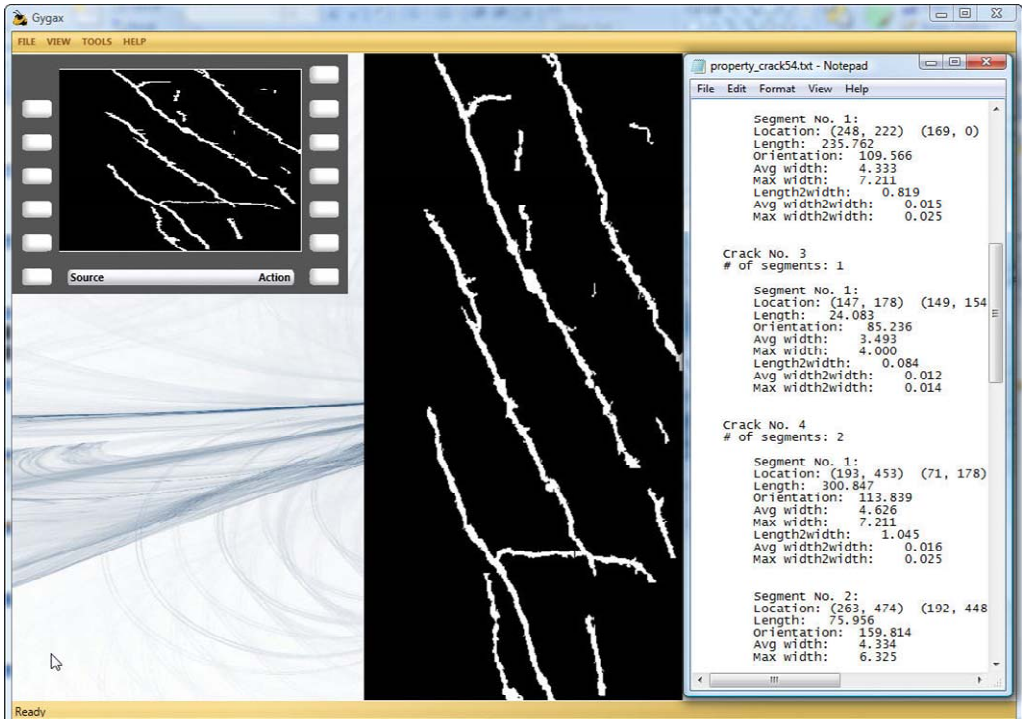
(a) Main interface of the prototype



(b) Image browsing



(c) Original image with cracks on the columns surface



(d) Crack map with retrieved properties

Figure 60: Screenshots of using the prototype to retrieve crack properties

4.5.2 Results on Crack Properties Retrieval

The images used to test the method of crack properties retrieval proposed in this research study are from the images and videos of the structured damaged by the 2010 Haiti earthquake (CEE News, 2010). Over one hundred images from the collected images and videos were used to validate the effectiveness of the method. The videos were decomposed into a sequence of images for the test purpose. The resolution of each image is fixed at 1600 x 1200. When collecting the images and videos of the damaged structures, the crack properties are also manually measured (Figure 61).

Figure 61: Damage information collection in Haiti

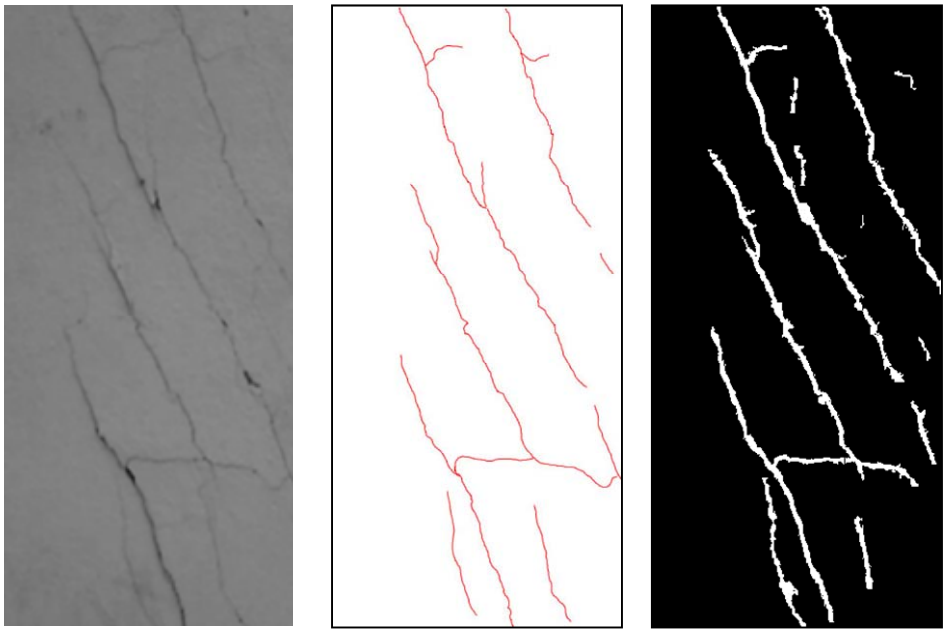
The performance of the method in crack detection is first measured. The crack detection part of the method is modified from the method proposed by Yamaguchi and Hashimoto (2009). The crack detection result is a crack map that represents all the crack

pixels inflicted on a structural member surface without any topological information. In order to measure the crack detection performance of the method, the evaluation is carried out by comparing the detected crack pixels with the cracks traced manually. The correctly detected crack pixels (true positive), the incorrectly detected crack pixels (false positive), and the undetected crack pixels (false negative) are identified with the procedure suggested by Iyer and Sinha (2006). One example of the matching procedure is illustrated in Figure 62, where Figure 62 (a) is the original image of a concrete column surface with cracks; Figure 62 (b) is the ground truth of the cracks on the surface traced manually; and Figure 62 (c) is the crack map produced by the method. The ground truth is dilated by a 5x5 structuring element (Iyer and Sinha, 2006). Compared with the dilated version of the ground truth, it can identify the crack pixels that are correctly detected (true positive) (Figure 62 (d)), the crack pixels that are incorrectly detected (false positive) (Figure 62 (e)), and the missed crack pixels (false negative) (Figure 62 (f)).

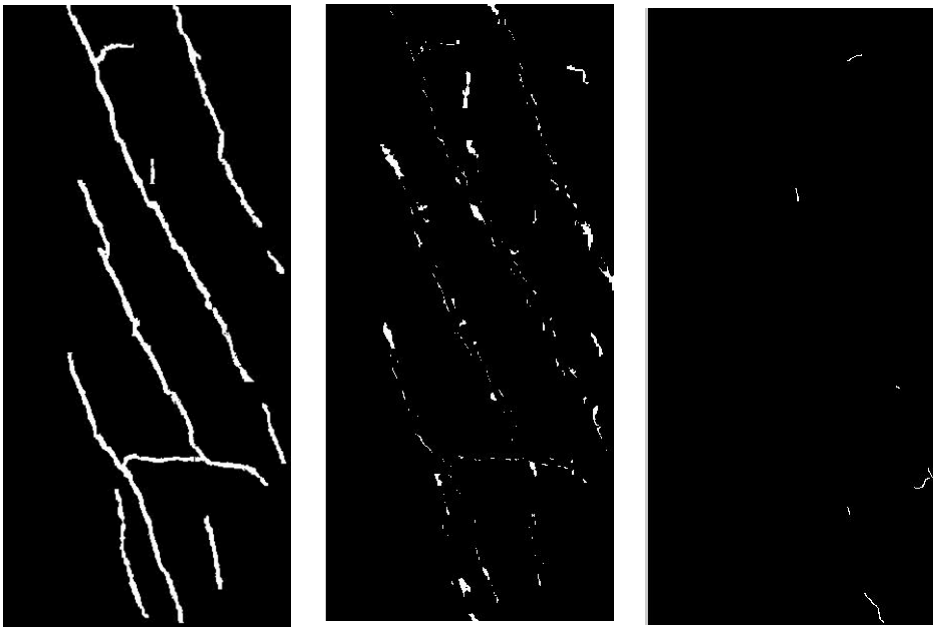
The crack detection precision is defined as the percentage of correctly extracted crack pixels from the detected results, and the recall is calculated as the percentage of real crack pixels that are detected. Table 4 illustrates the detection precision and recall calculated from all the test images.

Table 4: Crack detection precision and recall for 100 images

Summarization	
Total # of correctly detected crack pixels (TP):	13,0278
Total # of incorrectly detected crack pixels (FP):	72,599
Total # of real crack pixels not detected (FN):	11,594
Average precision (TP/(TP+FP)):	64.2%
Average recall (TP/(TP+FN)):	91.8%



(a) Original image (b) Ground truth (c) Crack map

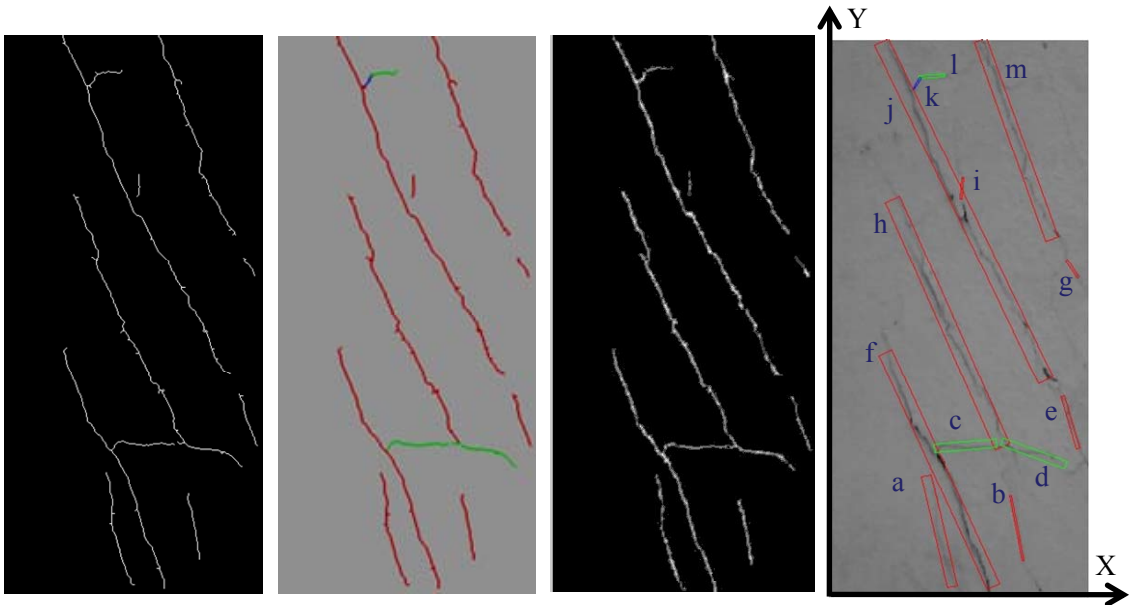


(d) True positive (e) False positive (f) False negative

Figure 62: A crack detection example

The performance of the method to retrieve crack properties is further measured on the cracks that are correctly detected. According to the previous tests, 225 cracks are

correctly detected by the method. The properties of these cracks are automatically measured. The properties include crack length, width, and orientation. The measurement of these properties is compared with the results of the manual measurements to determine the absolute measurement errors.



(a) Crack skeleton | (b) Segmented skeleton | (c) Distance map | (d) Crack segments

Figure 63: Intermediate results for crack properties measurement

Figure 63 shows one example of the intermediate results used by the method for automatic crack properties retrieval. The intermediate results include crack skeleton, skeleton segmentation, as well as the crack distance map. The corresponding measurement made by the method for this example is described in Table 5. Table 6 illustrates the statistical results of the crack properties measurement for all 225 cracks. The average errors in measuring the crack properties related to structural element dimensions are 3.29° for the crack orientation, 2.21% for the crack length and 0.35% for the maximum crack width. The results indicate that the relative measurements made by

the method are close to the ones from the manual surveys, although they are measured by image pixels.

Table 5: Measurement for 13 crack segments

	Crack Properties Measurement				
Crack Segment	Length - L (pixel)	Orientation	Max. Width (pixel)	L / W*	Max. W / W*
A	127.8	103°	6.3	0.444	0.022
B	73.0	101°	6.3	0.253	0.022
C	69.3	5°	5.7	0.240	0.020
D	76.0	160°	6.3	0.264	0.022
E	60.3	106°	5.7	0.209	0.020
F	287.7	115°	7.2	0.999	0.025
G	23.3	124°	7.2	0.081	0.025
H	300.8	114°	7.2	1.045	0.025
I	24.1	85°	4.0	0.084	0.014
J	418.1	115°	8.0	1.452	0.028
K	13.6	58°	6.0	0.047	0.021
L	29.1	4°	5.7	0.101	0.020
M	235.8	110°	7.2	0.819	0.025

Orientation – the angle to x-axis
L - crack length
Max. W - maximum crack width
W^* – Structural element width

Table 6: Measurement error for 224 cracks

	|Δ| - Orientation	|Δ| - L / W*	|Δ| - Max. W / W*
Total	739.65°	498.17%	78.66%
Average	3.29°	2.21%	0.35%
STD	2.70°	2.90%	0.49%

Orientation – the angle to x-axis
L - crack length
Max. W - maximum crack width
W^* – Structural element width

4.5.3 Discussion on Crack Detection and Properties Retrieval

The test results indicate that the measurement of the crack properties is matched to the manual measurement, as long as the crack pixels are correctly and fully retrieved by the crack detection method. However, none of the existing crack detection methods has been capable of retrieving all the crack pixels so far without an error, although they have no difficulty in finding cracks. Missing the detection of the crack pixels have a critical impact on the properties of crack retrieval and further on the crack-based safety evaluation of structural elements.

Even if all cracks can be correctly extracted, the measurement of crack properties may have little use for structural damage assessment, unless certain post-processing steps are performed. For example, the crack k and crack e in Table 5 are detected as two separate cracks. The two cracks may be considered as one crack from the perspective of a structural specialist, since they share the same orientation and are close enough in locations. This indicates a more serious structural problem. How to effectively combine the detected cracks needs to integrate the knowledge in structural damage assessment.

Also, the test results and previous studies (NCHRP, 2004) have shown that no matter how accurate existing crack detection methods are, there are always some cracks that are not visible in images, but are visible to human eyes. This limitation cannot be simply overcome by the improvement of crack detection methods or with higher resolution image capturing cameras. Other sensing devices need be considered. Fusing different types of sensing data to retrieve full detailed cracking information on structural members may be one possible way to solve this problem.

4.5.4 Implementation on Air Pockets and Discoloration Detection, Properties Retrieval, and Assessment

A prototype for the detection, properties retrieval, and assessment of air pockets and discoloration was developed using Microsoft Visual C++ and Open CV. Figure 64 shows the process of detecting and assessing air pockets and discoloration in a concrete surface image using the prototype. After a concrete surface image is loaded into the prototype, a user can choose to detect and assess air pockets or discoloration on the concrete surface separately. The prototype does not only locate air pockets or discoloration areas, but also calculate the corresponding visual impact ratios to assess the surface visual quality.

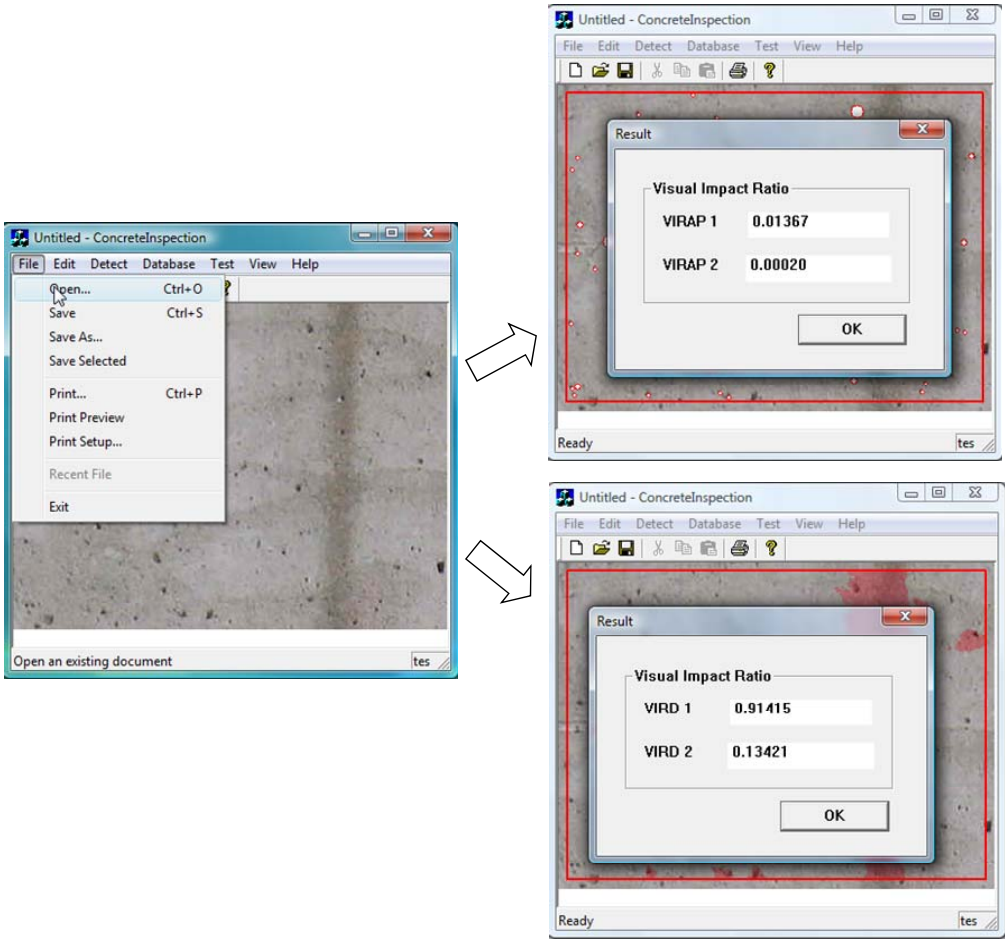


Figure 64: Process of detecting and assessing air pockets and discoloration

4.5.5 Results on Air Pockets Detection and Properties Retrieval

The precision of air pockets detection is defined as the percentage of the air pockets correctly detected in the total air pockets detected, while the recall for air pockets detection is the percentage of the air pockets correctly identified in the real air pockets. Typically, both ratios are useful for measuring the quality of detection results. The high precision means most air pockets are correctly detected, and the high recall means most real air pockets are detected.

The ratios can be used as the criteria to measure the preferable spot filter to be adopted for air pockets detection. Different filters produce different response values even when they are applied in the same image, as shown in Figure 52 (b) and Figure 52 (c). As a result, this affects their detection precision and recall ratios. Figure 65 shows the difference in the detection precision and recall of both spot filters. Although both filters can achieve the high precision ratios in their detecting results, the recall ratios of the filters are different. The filter composed with three concentric and symmetric Gaussian filters can retrieve 86% of real air pockets, while the filter composed with two concentric and symmetric Gaussian filters can only retrieve 74% of real air pockets. As a result, it is preferable to the spot filter formed with a weighted sum of three Gaussian filters.

In addition to choosing different filter types, the precision and recall can also be used to determine the optimum filter size. The filter size has an important impact on the detection results, since the high response of the air pockets to the filter is guaranteed only when the size of the air pockets is similar to the filter. In this research study, the filter with the size of 5x5, 7x7, 9x9 and 11x11 are tested. The relationship between the size of the filter and the detection precision and recall ratios is illustrated in Figure 66. The

precision ratios are maintained around 90%, while the recall ratios decrease when the size of the filter increases from 5x5 to 11x11 (83%, 66%, 53% and 43%). As a result, the filter size, 5x5, is selected.

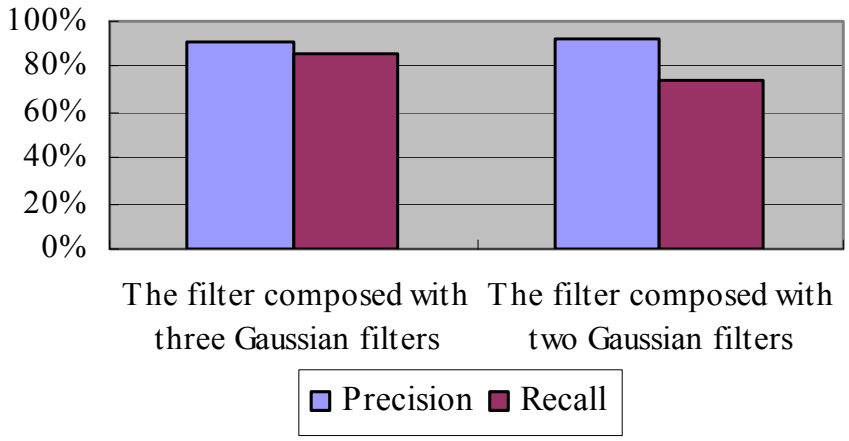


Figure 65: Precision and recall of air pockets detection with two filters

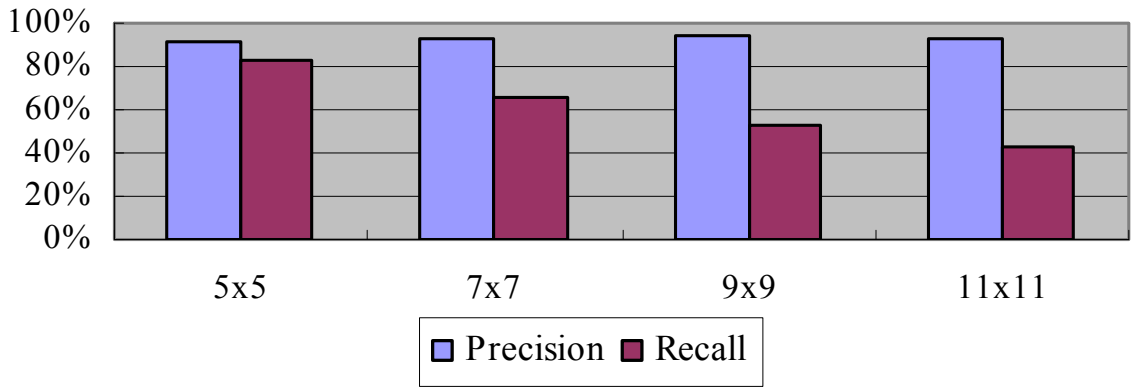


Figure 66: Precision and recall of air pockets detection for the filter with different size

The distribution of the air pockets in size is tested to measure the degree of approximating the size of real air pockets on a concrete surface. In doing so, a table is first manually created to classify real air pockets based on their diameters. Table 7 shows the classification result of the real air pockets on the concrete surface image, Figure 52 (a). The corresponding distribution of the air pockets is shown in Figure 67.

Table 7: Classification of air pockets according to their size

	The diameter of air pockets (X)										
	3>X	4>X ≥3	5>X ≥4	6>X ≥5	7>X ≥6	8>X ≥7	9>X ≥8	10>X ≥9	11>X ≥10	12>X ≥11	X≥12
# of air pockets	39	59	34	24	22	3	2	1	1	1	1

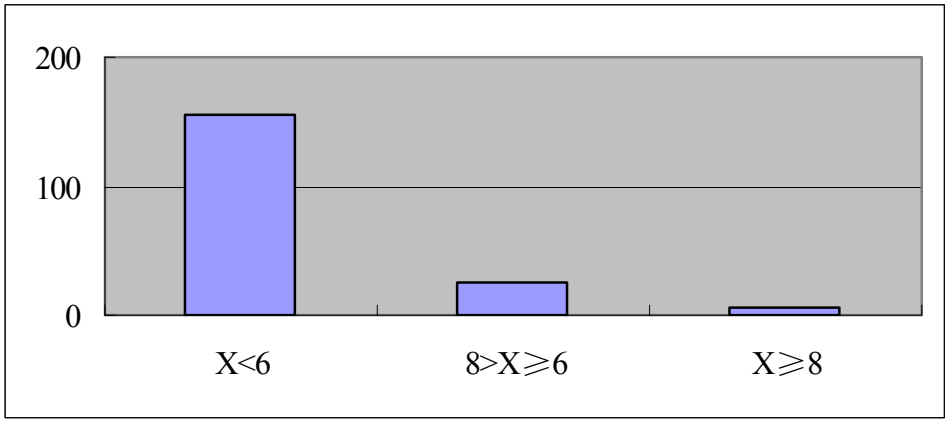


Figure 67: Air pockets distribution

Table 8 and 9 illustrate the effect of reducing the size of the concrete surface image (Figure 52 (a)) on the detection results of the air pockets in the image. The size of the image is first reduced according to the scaling percentages shown in Table 8. The selected spot filter is directly applied in each scaled version of the image. The air pockets detected by the filter are then recorded. In order to compare the detection ability of the filter on the air pockets with different size, all the detected air pockets are classified according to their diameters (Table 8). The recall ratio of the detected air pockets in each category is calculated (Table 9).

Table 8: The number of the air pockets detected in each size category

		The diameter of air pockets (X)										
		3>X	4>X ≥3	5>X ≥4	6>X ≥5	7>X ≥6	8>X ≥7	9>X ≥8	10>X ≥9	11>X ≥10	12>X ≥11	X≥12
Scaling percentage	100%	24	48	31	21	19	0	0	0	0	0	0
	90%	19	37	30	23	20	0	0	0	0	0	0
	80%	12	36	29	24	21	0	0	0	0	0	0
	70%	9	18	27	22	22	1	0	0	0	0	0
	60%	1	8	17	20	19	2	0	0	0	0	0
	50%	1	5	12	19	19	3	2	0	0	0	0
	40%	1	1	1	9	19	3	2	1	1	1	1
	30%	1	1	2	5	9	1	2	1	1	1	1
	20%	0	0	0	1	2	1	2	0	1	1	1
	10%	0	0	0	0	1	0	0	0	0	1	1

Table 9: The detection recall of the air pockets in each size category

		The diameter of air pockets (X)										
		3>X	4>X ≥3	5>X ≥4	6>X ≥5	7>X ≥6	8>X ≥7	9>X ≥8	10>X ≥9	11>X ≥10	12>X ≥11	X≥12
Scaling percentage	100%	62%	81%	91%	88%	86%	0%	0%	0%	0%	0%	0%
	90%	49%	63%	88%	96%	91%	0%	0%	0%	0%	0%	0%
	80%	31%	61%	85%	100%	95%	0%	0%	0%	0%	0%	0%
	70%	23%	31%	79%	92%	100%	33%	0%	0%	0%	0%	0%
	60%	3%	14%	50%	83%	86%	67%	0%	0%	0%	0%	0%
	50%	3%	8%	35%	79%	86%	100%	100%	0%	0%	0%	0%
	40%	3%	2%	3%	38%	86%	100%	100%	100%	100%	100%	100%
	30%	3%	2%	6%	21%	41%	33%	100%	100%	100%	100%	100%
	20%	0%	0%	0%	4%	9%	33%	100%	0%	100%	100%	100%
	10%	0%	0%	0%	0%	5%	0%	0%	0%	0%	100%	100%

According to the recall ratios, it proved that the filter's ability in air pockets detection can be maximized when the size of air pockets is close to the size of the filter. For

example, the size of the adopted filter is fixed at 5x5, in this experiment. The filter can detect only 86% of the air pockets with the size range between 6 and 7, but can detect 91% of the air pockets with the size range between 4 and 5 in the original image. This point can also be illustrated in the other way. Still take the air pockets with the size range between 6 and 7 for an example. When the image size is reduced, the number of the detected air pockets in this size range (between 6 and 7) is increased. This is because the reduction of the size of the image makes large air pockets become small ones. The maximum detection ability for the air pockets in this size range is achieved when the image size is reduced to 70%. At that moment, the size of air pockets is actually scaled down to the range between 4.2 (6*0.7) and 4.9 (7*0.7), which is very close to the size of the filter (5x5).

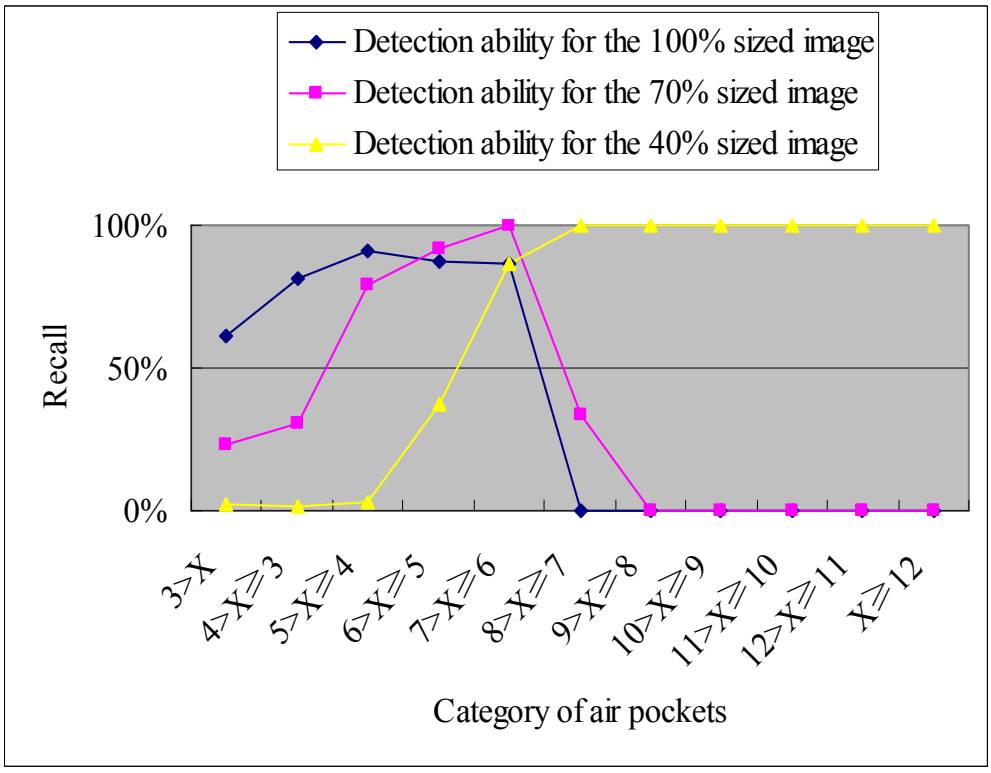


Figure 68: Detection recall of the air pockets in three scaling images

When the detection recall ratio of the filter in each air pocket category is retrieved, the detection ability of the filter can be measured and depicted as a curve as shown in Figure 68. In the original image (100% sized image), the filter can mainly detect small air pockets but miss large ones. This limitation is overcome by adjusting the size of the image. For example, when the size of the image is reduced to 40% of the original image, the large air pockets can be detected, but the small air pockets are missed.

In order to detect as many air pockets as possible, three scaling percentages are selected. They are 100%, 70%, and 40%. The detection results using the three scaling percentages are illustrated in Figure 69, where the detected air pockets are represented by red circles. The larger an air pocket, the bigger a red circle.

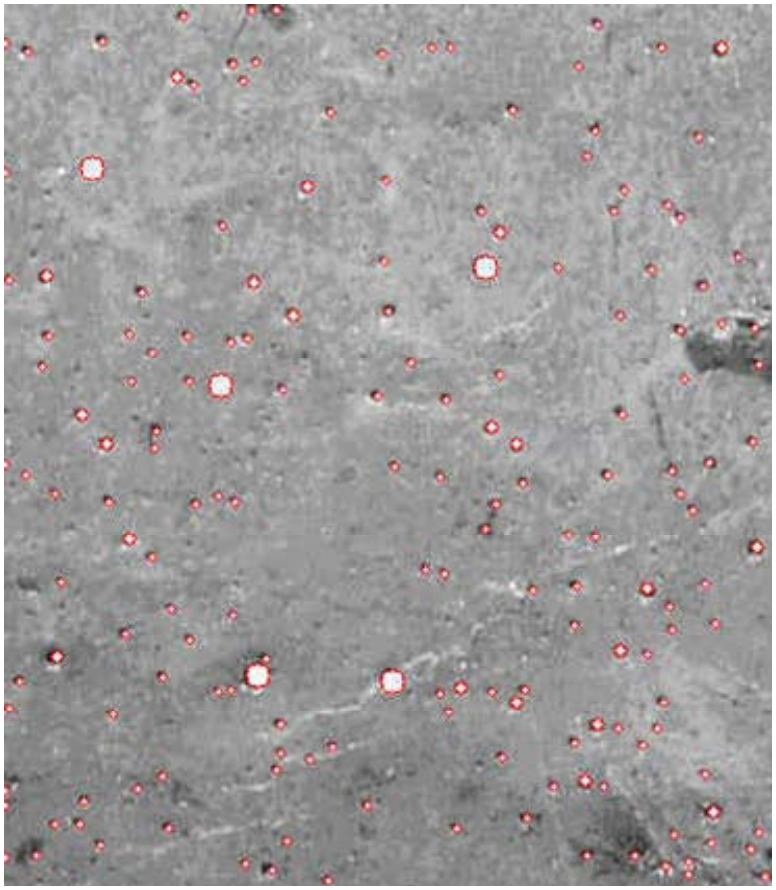


Figure 69: Air pockets detection through spot filtering and image scaling

The detected air pockets are classified into three levels corresponding to the scaling percentages. The number of the detected air pockets in these three levels is recorded in Figure 70. Compared with the number of the real air pockets in the three levels, it can be seen that the distribution of the detected air pockets is almost matched with the distribution of the real air pockets.

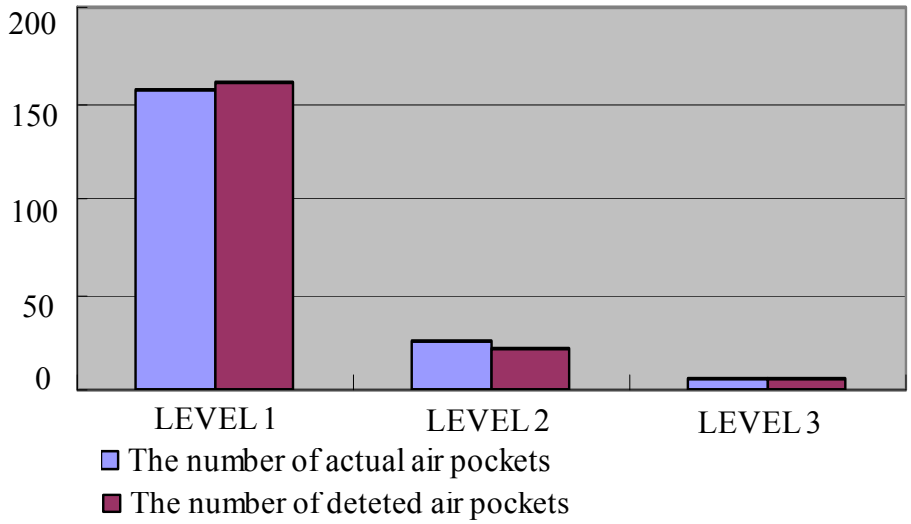


Figure 70: Distributions of detected air pockets and actual air pockets

A database of concrete surface images is tested in the prototype. One part of test results is shown in Table 10, where the results are measured using two ratios (precision and recall). According to Table 10, it can be seen that the detection precision ranges from 77.4% to 96.8% and the average detection precision reaches 91.1% with the standard deviation of 5.5%. The detection recall ranges from 77.0% to 92.4% and the average detection recall reaches 85.6% with the standard deviation of 6.0%.

Table 10: Detection precision and recall

Image	Detected air pockets	Real air pockets	Correctly detected air pockets	Precision	Recall	VIRAP1	VIRAP2
(a)	188	190	163	86.7%	85.8%	2%	0.10‰
(b)	132	136	124	93.9%	91.2%	3%	0.21‰
(c)	139	159	123	88.5%	77.4%	3%	0.24‰
(d)	95	106	90	94.7%	84.9%	2%	0.18‰
(e)	93	113	87	93.5%	77.0%	4%	0.42‰
(f)	126	154	122	96.8%	79.2%	3%	0.20‰
(g)	195	226	184	94.4%	81.4%	2%	0.12‰
(h)	31	26	24	77.4%	92.3%	2%	0.82‰
(i)	181	184	170	93.9%	92.4%	6%	0.31‰
(j)	120	118	107	89.2%	90.7%	4%	0.35‰
(k)	173	180	160	92.5%	88.9%	4%	0.21‰
Average in (a) – (k)				91.1%	85.6%		
Standard deviation in (a) – (k)				5.5%	6.0%		

Precision - # of air pockets correctly detected / # of air pockets detected
Recall - # of air pockets correctly detected / # of actual air pockets

At the same time, the visual impact ratios of air pockets are calculated. The visual impact ratios can be used to measure the visual quality of concrete surfaces. For example, although both concrete surface image (e) (Figure 71 (a)) and image (g) (Figure 71 (b)) in Table 10 are covered with many air pockets, comparatively, the visual quality of concrete surface image (g) is better than the image (e), because the concrete surface image (e) mainly consists of the air pockets with large size while the concrete surface image (g) mainly consists of the air pockets with small size. This point is also reflected based on their visual impact ratios. Both VIRAP1 and VIRAP2 of the concrete surface image (e) are higher than those of the concrete surface image (g) in Table 10.

(a) Image (e) in Table 10

(b) Image (g) in Table 10

Figure 71: Image (e) and image (g) in Table 10

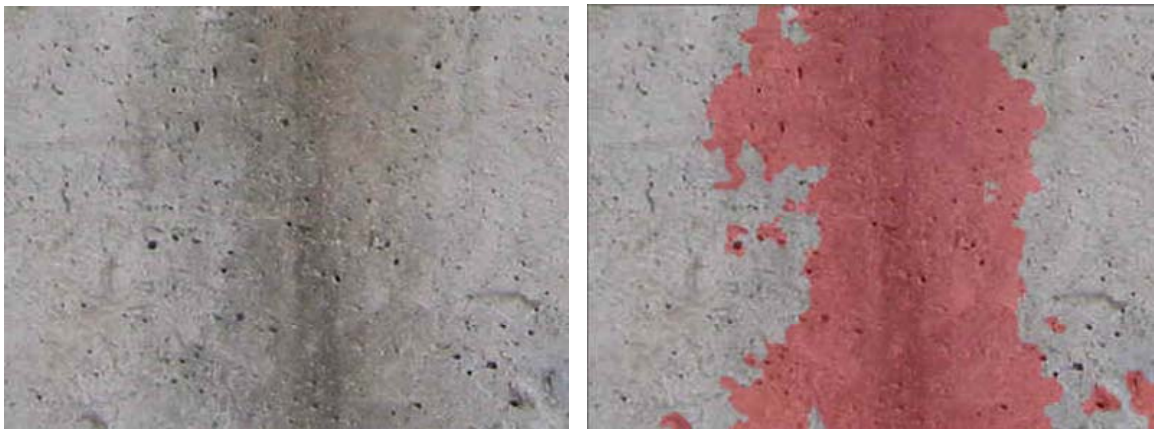
The effectiveness of the method in this research study is compared with the method presented by Suwwanakarn et al (2007) using the image in Figure 52 (a) as an example. Two cases are studied in the air pockets detection. The first one is the application of the 11x11 filter and 5x5 filter type 1, while the second one is the application of 11x11 filter and 5x5 filter type 2. In the former case, the detection precision (86.0%) almost equals to the detection precision using the method presented in this paper (86.7%), but the corresponding recall (48.4%) is lower than the detection recall using the method in this research study (85.8%). In the latter case, both detection precision and recall (69.2% and 38.9%) are lower than the detection precision and recall using the method presented in this paper. This point has been illustrated in Table 11.

Table 11: Methods comparison

Air pockets	The method of Suwwanakarn et al (2007)		The method in this research
	11x11 filter + 5x5 filter type 1	11x11 filter + 5x5 filter type 2	
Real	190	190	190
Detected	107	107	188
Correctly detected	92	74	163
Precision	86.0%	69.2%	86.7%
Recall	48.4%	38.9%	85.8%

4.5.6 Results on Discoloration Detection and Properties retrieval

Similar to air pockets detection, a database of concrete surface images was used to test the method of discoloration detection proposed in this research study. One discoloration example is shown in Figure 72. The discoloration areas are marked red. Compared with the original concrete surface image, it can be seen that the existing discoloration regions in the original image can be successfully identified.



(a) (b)

Figure 72: Discoloration detection

4.5.7 Assessment of Air Pockets and Discoloration

In order to qualitatively evaluate the visual quality of concrete surfaces, on one hand, manual inspection for concrete surface images is performed. Figure 73 shows the manual inspection results from an experienced inspector. The inspector rated each concrete surface image sample according to his perceived quality in terms of air pockets and discoloration.

The higher the rating is made, the poorer the visual quality of concrete surfaces is. The lower the rating point the inspector gave, the better the quality of concrete surface is. For example, in Figure 73, the inspector rated the surface quality of concrete sample 3

and 5 in terms of air pockets lower than he rated the surface quality of concrete sample 4 and 6, since the size of the air pockets in the concrete surface sample 3 and 5 is smaller than that of the air pockets in the sample 4 and 6. Also, the inspector would decide whether the quality of the concrete surface in an image is acceptable or not based on the rating points.

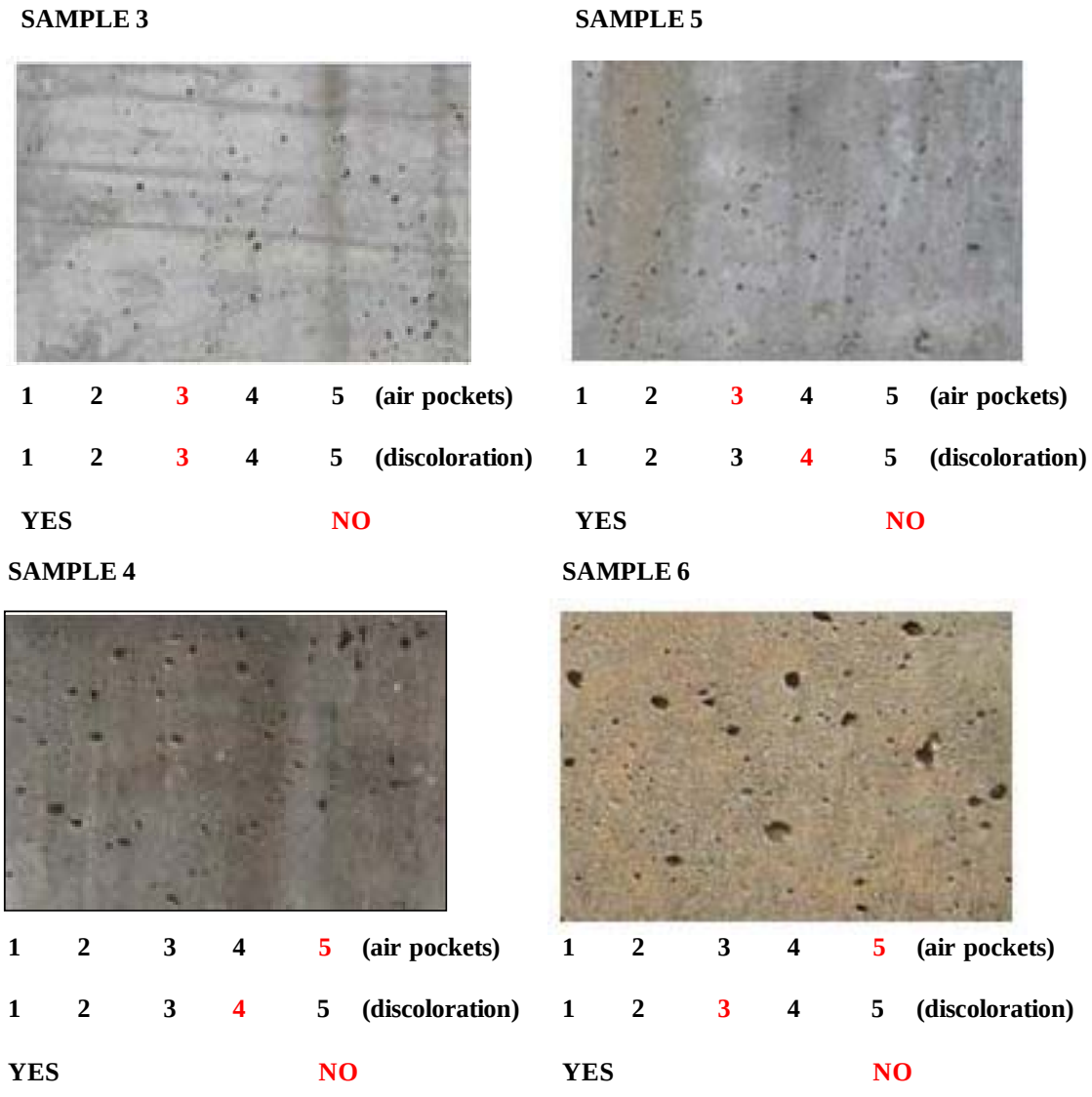


Figure 73: Manual inspection results

On the other hand, the visual impact ratios of air pockets (VIRAP1 and VIRAP2) and discoloration (VIRD1 and VIRD2) for the same concrete surface images are calculated.

One part of the ratios is illustrated in Table 12 and 13. Table 12 shows the relationship between the manual ratings and the VIRAP1 and VIRAP2. According to the manual rating results, it is found that the smaller/larger the VIRAP1, the lower/higher the rating of concrete surface images, which means the better/poorer the concrete surface quality in terms of air pockets. The VIRAP2 can be used as an auxiliary indicator for the VIRAP 1. When two concrete surface images have the same VIRAP1, the quality of the concrete surface that has small VIRAP2 is better than that of the surface that has large VIRAP2. VIRAP2 itself, however, cannot be used to measure the quality of concrete surfaces. For example, in Table 12, although the VIRAP2 of the image 8 is larger than those of other images, the quality of the image 8 in terms of air pockets is one of the best. It is because VIRAP2 is calculated as the VIRAP1 divided by the number of the air pockets detected. In the image 8, the number of the air pockets is low, compared with the number of the air pockets detected in other images. This makes the VIRAP2 of the image 8 larger than those of other images. Compared with the manual ratings, the threshold for the VIRAP1 was set to 0.02. When the VIRAP1 of a concrete surface image is below 0.02, the quality of concrete surfaces in terms of air pockets is acceptable. Else, it is unacceptable.

Table 13 shows the relationship between the manual ratings and the VIRD1 and VIRD2. According to the manual rating results, it is found the quality of concrete surface in terms of discoloration can be high, only when VIRD1 and VIRD2 are both small. For example, in Table 13, although the image 6 has a higher VIRD1 than the image 5, its lower VIRD2 makes the image 6better than the image 5. As a contrast, even if the image 1 has a larger VIRD2 than the image 5, the image 1 is still better than the image 5, since the VIRD1 of the image 1 is smaller than the image 5. In order to measure the surface

quality of concrete in terms of discoloration, the threshold for VIRD1 is set at 0.7, and the threshold for VIRD2 is set at 0.2. When both VIRD1 and VIRD2 are below the corresponding threshold values, the surface quality of concrete in terms of discoloration is acceptable. Else, it is unacceptable.

Table 12: Relationship between the manual ratings and the VIRAP1 and VIRAP2

Image	VIRAP1	VIRAP2	Manual Rating
1	0.02696	0.00021	4
2	0.03305	0.00024	4
3	0.0162	0.00017	3
4	0.03679	0.00042	5
5	0.0209	0.00012	3
6	0.04082	0.00034	5
7	0.00623	0.00026	1
8	0.00633	0.00063	1
9	0.02235	0.00025	4
10	0.023	0.00027	5

Note: 1 is the best and 5 is the worst in manual rating

Table 13: Relationship between the manual ratings and the VIRD1 and VIRD2

Image	VIRD1	VIRD2	Manual Ratings
1	0.32395	0.16009	2
2	0.23444	0.0934	2
3	0.43901	0.18663	3
4	1.03608	0.44199	4
5	0.71103	0.21871	4
6	1.99974	0.02433	3
7	0.67331	0.08509	2
8	2.37998	0.49343	4
9	1.14605	0.48614	3
10	1.16442	0.36918	4

Note: 1 is the best and 5 is the worst in manual ratings

To test the effectiveness of the selected threshold values in evaluating the visual quality of concrete surfaces, the images in the database are further qualitatively evaluated using the visual impact ratios and the threshold values. One part of the inspection results are shown in Table 14. It can be seen that the automated inspection results are same as the results from manual inspection in most cases.

Table 14: Automated inspection results and manual inspection results

Image	Automated inspection	Manual inspection
1	Not acceptable	Not acceptable
2	Not acceptable	Not acceptable
3	Acceptable	Not acceptable
4	Not acceptable	Not acceptable
5	Not acceptable	Not acceptable
6	Not acceptable	Not acceptable
7	Acceptable	Acceptable
8	Not acceptable	Not acceptable
9	Not acceptable	Not acceptable
10	Not acceptable	Not acceptable

4.5.8 Discussion on Air Pockets and Discoloration Detection, Properties Retrieval, and Assessment

The resolution of the images used for the detection, properties retrieval, and assessment of air pockets and discoloration in this research study is fixed at 1024 by 768. The effectiveness of the critical parameters and threshold values in the method, such as the image scaling percentages and the threshold value for the visual impact ratios, have been tested using the concrete surface images under this resolution. If the images with other resolutions are used, the critical parameters and the threshold values have to be re-calculated following the same process described before.

The impact of light conditions on the effectiveness of the threshold values is significant only when the light conditions are extreme. The images tested in this research study were taken at different natural sunlight conditions. According to the test results, the threshold values were effective as long as air pockets and discoloration were visible. However, the threshold values were affected in some special cases. For example, if an image was really dark, the near-black discoloration on the concrete surface was difficult to detect. The calculated VIRD1 and VIRD2 were smaller than its true VIRD1 and VIRD2, which made the automatic rating inconsistent with the manual rating. Therefore, it is the responsibility of a user to provide the images without extreme light effects. The images with high contrast and/or dark shadows at concrete surfaces could not be used.

4.6 Summary

This chapter described two novel methods in defects/damage detection, properties retrieval, and evaluation. The first method is about retrieving the properties of the cracks on a concrete column surface. The method starts with a percolation-based crack detection method to locate crack points on each concrete structural element surface. Then, the crack properties, such as length, orientation and width, are retrieved from the topological skeletons of cracks and the distance field of crack pixels in the map. The properties are further related to the dimension and orientation of the structural element to produce relative measurements.

The method was implemented into a prototype developed by Microsoft Visual Studio .NET. Over 100 structural member images were used to test the performance of the method. These images were collected from structures damaged by the January 2010 earthquake in Haiti. The crack properties measured by the method were compared with

manual measurements, and the absolute and relative errors in the measurement of length, orientation and width are calculated. The average measurement error (3.29° for orientation, 2.21% for relative crack length and 0.35% for relative maximum crack width) indicated that the proposed method can automatically and correctly retrieve crack length, orientation and width.

The second method is about the detection, properties retrieval, and evaluation of air pockets and discoloration on a concrete surface. In the method, the air pockets are detected by the application of a spot filter in an image pyramid. The discoloration areas are identified using image segmentation and region comparison. The properties of the air pocket and discoloration are then retrieved. Their visual impact ratios are calculated. The quantitative visual impact ratios are compared with the manual ratings to select appropriate threshold for each visual impact ratio. This way, the link between the visual impact ratios and the surface quality of concrete is established. The visual quality of a concrete surface can be evaluated using the quantitative visual impact ratios of air pockets and discoloration.

The method presented was implemented using Microsoft Visual C++ and Open CV. A database of concrete surface images containing air pockets and/or discoloration surface were used to test methods. The results validated the effectiveness of the method. According to test results, air pockets and discoloration can be successfully identified. The visual impact ratios for air pockets and discoloration can objectively and accurately evaluate the visual quality of concrete surfaces in terms of air pockets and discoloration.

CHAPTER 5
POTENTIAL APPLICATION AREAS

This chapter presents the potential application areas using the results from this research. The research is focused on the recognition of concrete columns from images and videos, and the detection, properties retrieval, and evaluation of defects and damage, such as cracks, air pockets, and discoloration inflicted on concrete surfaces. The results from this research are expected to automate and facilitate many applications, including rapid post-disaster building evaluation, routine civil infrastructure inspection, project progress monitoring, productivity measurement, etc.

5.1 Rapid Safety Evaluation of Buildings for Emergency Responders

When an earthquake happens, people may be trapped in damaged buildings. They are waiting for emergency responders, such as fire fighters, to come and save them. When emergency responders arrive, they cannot enter a damaged building to start life search and rescue tasks immediately. The first thing that they need to make sure is the building is safe for them to enter. They do not want to become victims, when they just go into a building and the building collapses.

Since emergency responders must act within a few hours of an earthquake event, a rapid safety evaluation through a lengthy and detailed structural assessment based on design drawings and non-destructive tests is not feasible (NIST, 2005; Kwan and Lee, 2005). Instead, only a crude yet rapid visual safety assessment is possible. This visual evaluation is the current standard practice, and is performed by a trained structural specialist, if one is available and has been dispatched to the site.

However, there are not enough qualified structural specialists available for each emergency response team. So far, each national urban search and rescue task forces has been teamed with one dedicated structural specialist, but local emergency response teams, which are responsible for saving more than 75% of disaster victims during initial response, do not have a structural specialist working with them dedicatedly (FEMA, 2009). Even in those cases where a structural specialist is on-site, a lot of manual evaluation time is necessary when hundreds or even thousands of buildings have been damaged. The adverse effect is that the survival rate of the trapped victims drops significantly. From Table 15, it can be seen that over 91% of people trapped in collapsed structures can survive if they are rescued within 30 minutes; however, this value is reduced to 7.4% once trapped for five days (UKFSSART, 2007).

Table 15: Chance of survival of victims (data from UKFSSART (2007))

Survival rate	Duration of entrapment
91.0%	30 minutes
36.7%	48 hours
7.4%	120 hours

My research can automatically get the defect/damage information on the concrete columns of a building. Therefore, it can be used to provide a new and automated solution to evaluate the safety of entry into a damaged building for emergency responders. As illustrated in Figure 74, a portable video camera is attached on each emergency responder's hardhat. When emergency responders walk into a damaged building, the videos collected by the cameras will be analyzed to recognize concrete columns, detect their inflicted damage, and calculate the corresponding damage properties. This information can be further transmitted outside through a local wireless network and via

the Internet to structural specialists at remote sites. The specialists will watch the videos with the damaged properties provided almost in real time. When they perceive the potential risk of a building collapse, the specialists can inform the emergency responders to evacuate before the collapse happens. This means, if there is an earthquake in California, the structural specialists in Georgia can also be involved in the evaluation process building safety, which overcomes the limitation of not having enough qualified structural specialists in a local area. Also, the structural specialists do not have to manually measure the damage properties, which greatly speed up the evaluation process. Compared with other non-destructive evaluation test devices, the video cameras in this solution are definitely mobile and affordable.

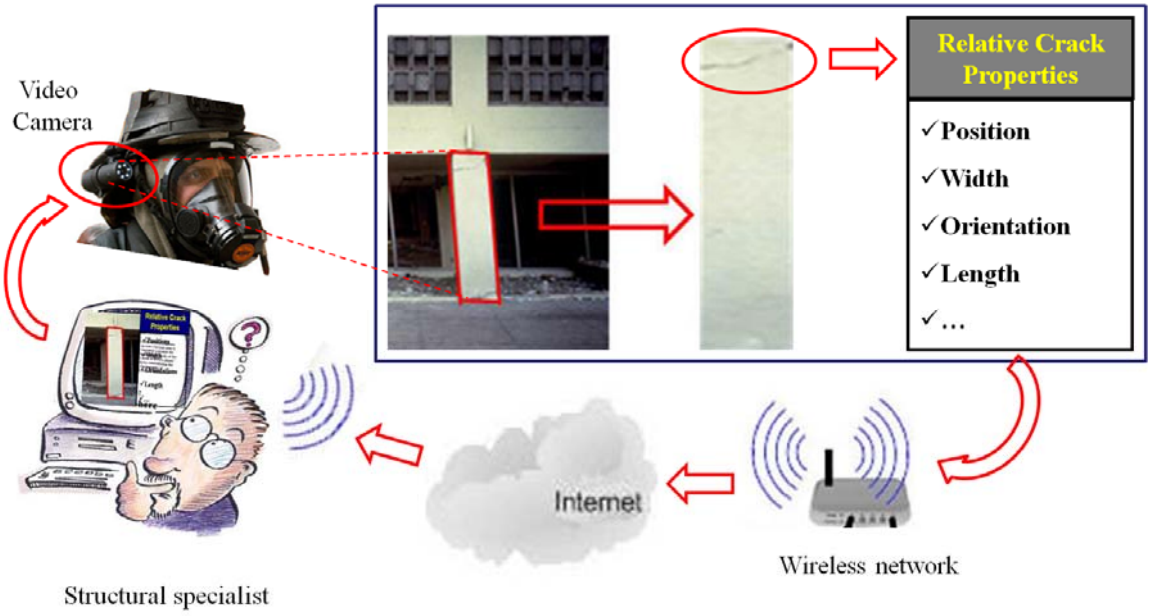


Figure 74: Rapid building safety evaluation for protecting emergency responders

5.2 Rapid Loss Estimation for Post-earthquake Structures

The structures damaged by earthquakes need to be repaired as fast as possible, in order to minmize the economic loss and disruptions induced by the earthquakes. However, the

lack of available inspectors, when combined with the large volume of inspection work, makes this procedure time-consuming. Even for performing the required work of post-earthquake structure inspection, it may take weeks or months to complete, which brings adverse economic and societal impacts on the affected population.

Prompted by the critical role of post-earthquake inspection in hazard mitigation and the need for its fast performance in earthquake damaged areas, it is necessary to automate the procedures of estimating post-earthquake building repair cost and time. My research can failicate this procedure in the area of automatically collecting structural quantitative damage data. The collected damage data are expected to be used to provide quantitative assessment and estimate associated building repair cost and time.

Specifically, a high-resolution video camera can be attached on the hard-hat of an evaluator (Figure 75). The video frames collected by the camera are transmitted via a wireless enabled PDA mounted on the evaluator's outfit to a computer off-site for analysis. There, critical structural elements, like concrete columns, in each frame are searched with the help of the novel column detection recognition method proposed in this research. The damage inflicted on these concrete columns (mainly visible in the form of cracks, concrete spalling and reinforcement buckling) is then detected using state-of-the-art detection techniques. The spatial properties of the detected damage are then measured and superimposed on the detected concrete column to measure relative damage properties (e.g. length, width, orientation and position of cracks) in relation to the column's dimensions and orientation, so that the column's load bearing capacity can be approximated as a damage index. In parallel to this process, the building structural type and the columns arrangement per floor (damaged and undamaged) is recorded by the user

while performing the building safety evaluation. The collected information (building type, damage index, and columns' arrangement) is then used to query a fragility database. This database is constructed from analyses of existing and on-going experimental data, and will contain fragility curves of buildings that report the probability of various levels of structural damage given the structure type, the columns' arrangement, and the detected damage in the columns. By consulting these curves, the query estimates the probability of being in different damage states and the corresponding repair cost. The query results can be reported to the evaluator, who can then use it to make an informed decision.

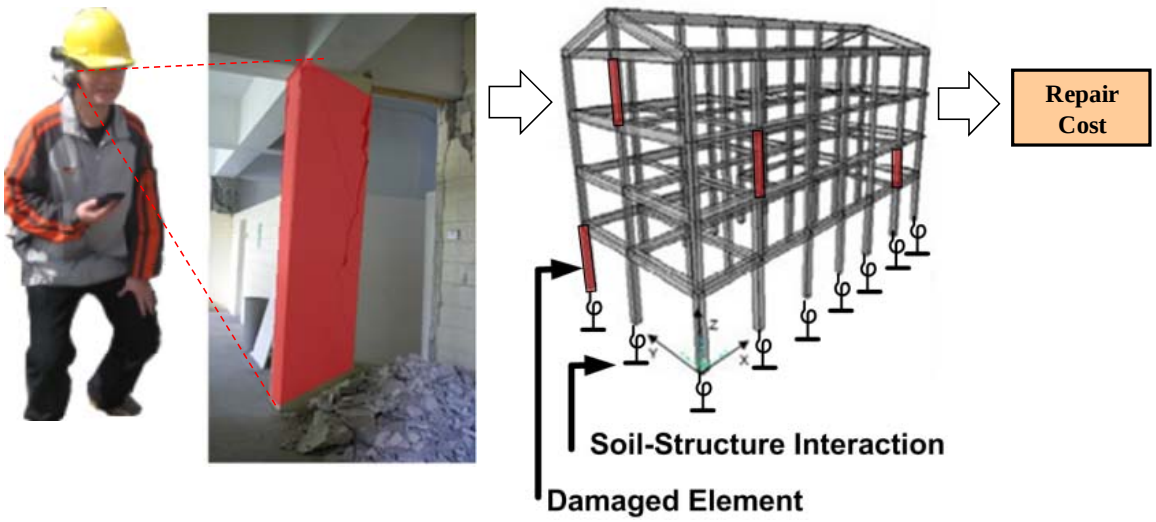


Figure 75: Rapid loss estimation for post-earthquake structures

5.3 Civil Infrastructure Assessment and Modernization

It is no secret that large portions of old existing civil infrastructure in the United States (e.g. 600,000 highway bridges, and 4,000,000 miles of public roadways) are aging and require significant maintenance and expansion in structural integrity and sustainability. Considering limited budgets available, state or city agencies need to develop

comprehensive but prioritized plans to convert this substantial inventory of large, dated civil infrastructure into sustainable, energy efficient structures.

However, the plan is difficult to make wisely with the current practice of civil infrastructure assessment. Today, the most common form of assessing civil infrastructure is visual inspection supplemented with other qualitative methods. The inspection process is time-consuming, and results are subjective. Once inspection determines a defective element, a decision is made to repair, retrofit, or replace the compromised element. Its impact on the civil infrastructure as a system is not adequately evaluated. The result is that the corresponding maintenance decisions are not always wise. For example, a steel reinforcement project was scheduled to strengthen the I-35W Mississippi River Bridge, but it was revealed later that the drilling for the retrofitting in fact weakened the bridge. The bridge was scheduled to repair in 2020, but it collapsed in 2007.

My research can enhance the current visual inspection practice. Take routine highway bridge inspection for an example. Inspectors or engineers can assess a bridge with a video camera and a laptop (Figure 76). The camera collects the accurate and detailed video data of the bridge. These video data can be used to generate the geometric object model of the bridge using videogrammetry techniques. The bridge columns and other elements in the model are then recognized by searching the captured video frames. In parallel, the visible damage and defects (e.g. cracking and staining) on these elements are detected and their properties are retrieved. The retrieved properties are evaluated based on the elements' condition state descriptions specified in a bridge management system. The determined condition state of each element can be further integrated into the bridge geometric model for bridge condition recording and assessment purposes. This way, any decision in

repairing, retrofitting, or replacing a single bridge element can be evaluated systematically.

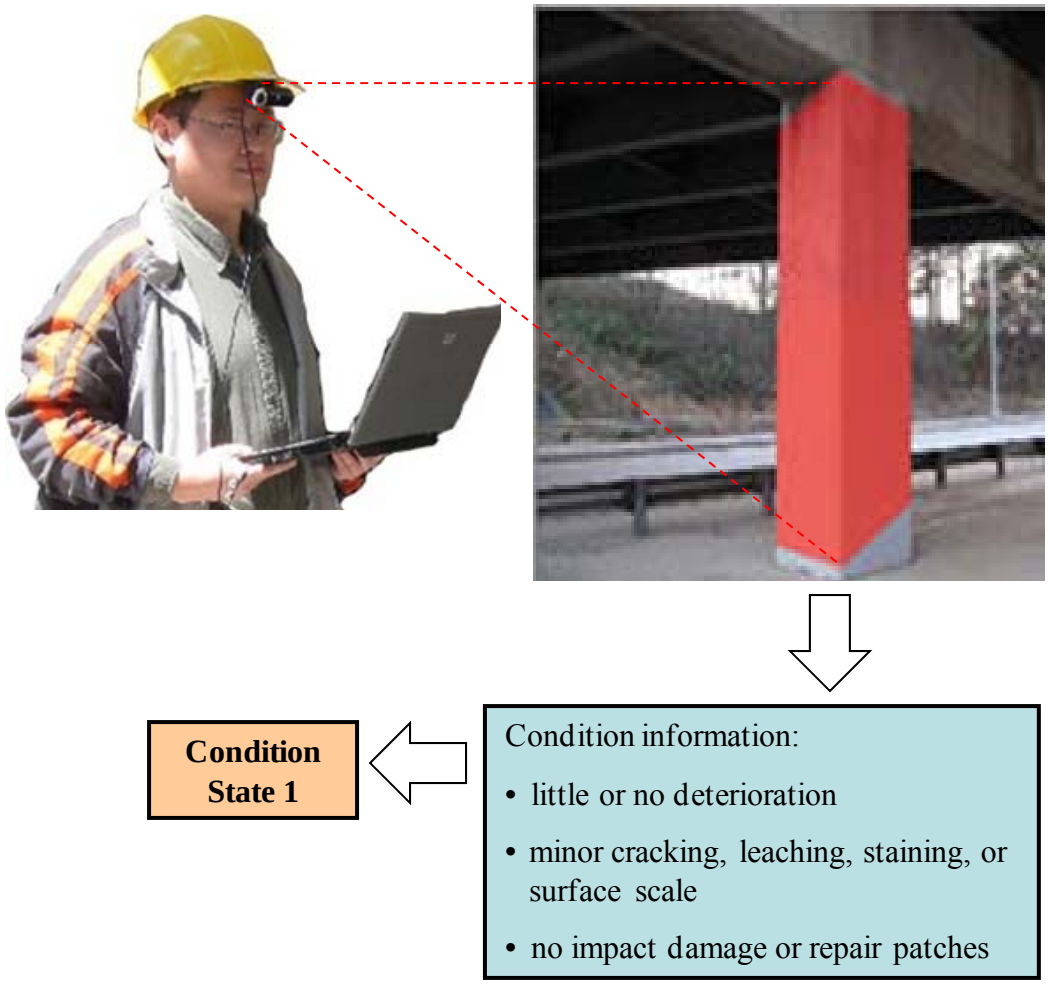


Figure 76: Automated bridge column inspection within the POINTS rating system

My research can alleviate subjective nature of the current visual inspection practice by reducing the reliance of inspection results on inspectors' personal experience and knowledge. On top of that, my research is also expected to considerably reduce the required time and work in civil infrastructure assessment, and to provide detailed and accurate condition information of civil infrastructure for infrastructure performance analysis. Moreover, it can be used to evaluate current infrastructure design and construction processes in addressing structure durability and affordability issues in future.

5.4 As-built Infrastructure Modeling

As-built infrastructure modeling is the process of generating digital 3D models of civil infrastructure. It starts from collecting infrastructure's spatial data as a series of 3D point coordinates using remote sensing techniques (Ikeuchi and Sato, 2001), and then transforming these data into structured or object-oriented representations like CAD models (Remondino and El-Hakim, 2006). The 3D models of civil infrastructure have been proved useful at various stages of the infrastructure life cycle. For example, an accurate and detailed 3D model of the built environment can be used to identify existing defects in construction by comparing the measurements taken from the as-built model and the measurements taken from the design (Gordon et al. 2003). Also, the accurate and fast 3D models of site facilities can be adopted to perform efficient layout planning of a construction site to optimize facility running cost and/or time (Ma et al. 2005).

However, 3D as-built models are not widely produced in most projects because a lot of effort is necessary to manually convert remote sensing data from photogrammetry or laser scanning to an as-built model. This process is human dependent to a great extent, and therefore, time-consuming and costly. Previous research studies indicated that over two thirds of the effort is spent on manually converting surface data to a 3D model even when modeling simple infrastructure (Sternberg et al. 2004; Jaselskis et al. 2005). Similar findings were also observed in one of the author's projects in comparing different optical sensor based spatial data collection techniques (Zhu and Brilakis, 2009).

This research can facilitate the generation of as-built infrastructure models in two aspects. First, it automatically retrieves civil infrastructure element related information from images. The research provides examples of how to create visual pattern recognition

(VPR) templates that can recognize infrastructure-related elements, such as concrete columns and the defects/damage inflicted on structures, based on their visual features. Based on the findings in this research, the VPR templates for other types of civil infrastructure elements can also be correspondingly created. This way, the magnitude of the demands of manually retrieving civil infrastructure element information can be significantly diminished in the processes of infrastructure modeling.

Second, the infrastructure element related information retrieved by the research can be used to automatically classify 3D points. Today, it is feasible to get 3D points from images with existing 3D reconstruction techniques. The difficulty lies in how to classify these 3D points into an information-rich, object-oriented as-built model. My research alleviates this difficulty. For example, if concrete columns are recognized in images, 3D point clouds that belong to the concrete columns can be easily classified according to the mapping from image pixels to 3D points (Figure 77).

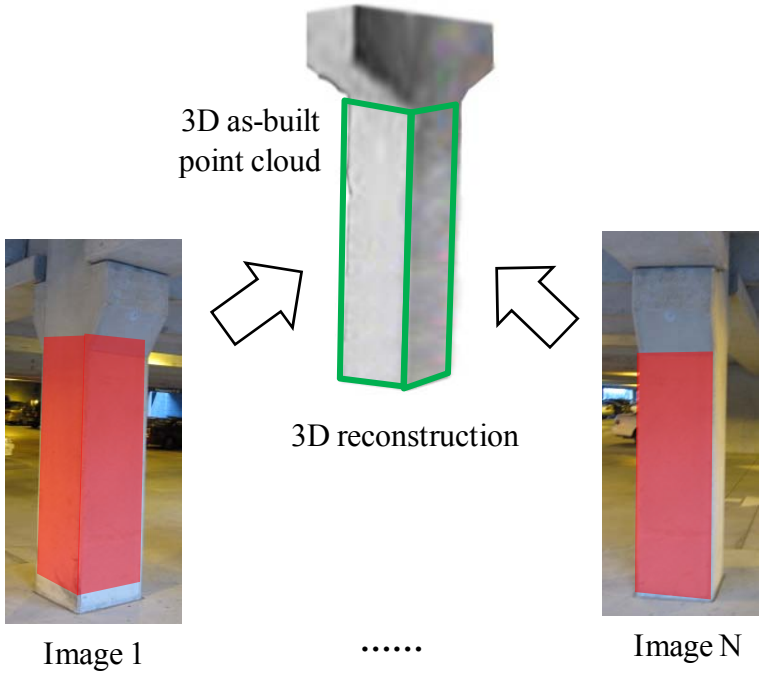


Figure 77: Automated 3D as-built modeling

5.5 Progress Monitoring at the Element Level

It is necessary to monitor the construction progress of a project to make sure the project can be completed within its schedule. Any delays in the schedule may lead to the huge loss of project profits. The current practices in construction progress monitoring require project managers and/or site engineers to spend a significant amount of time in collecting project progress information at a construction site from construction drawings, schedules, and field reports. Even so, it is still difficult to guarantee that the collected information can be used to accurately report the project progress, since the site engineers always collect the information based on their interpretation of what needs to be measured and how it needs to be measured (Golparvar-Fard et al. 2009).

My research can be used to help project managers and/or site engineers automatically extract useful structural element as-built information from digital image. This as-built information can be referred to the project's as-planned model to figure out whether the project is behind or ahead the schedule (Figure 75). For example, if eleven concrete columns are required to be constructed according to the schedule but only nine concrete columns have been recognized from as-built images, then it is easy for project manager and/or site engineers to tell the construction activity of concrete column pouring has led to the delay in the project. The early detection of actual schedule delay at structural element level will provide the project mangers and/or site engineers with an opportunity to immediately initiate remedial actions to minimize the impact of the delays. Also, it will help the project mangers and/or site engineers review when, why, and how the project delays happened.

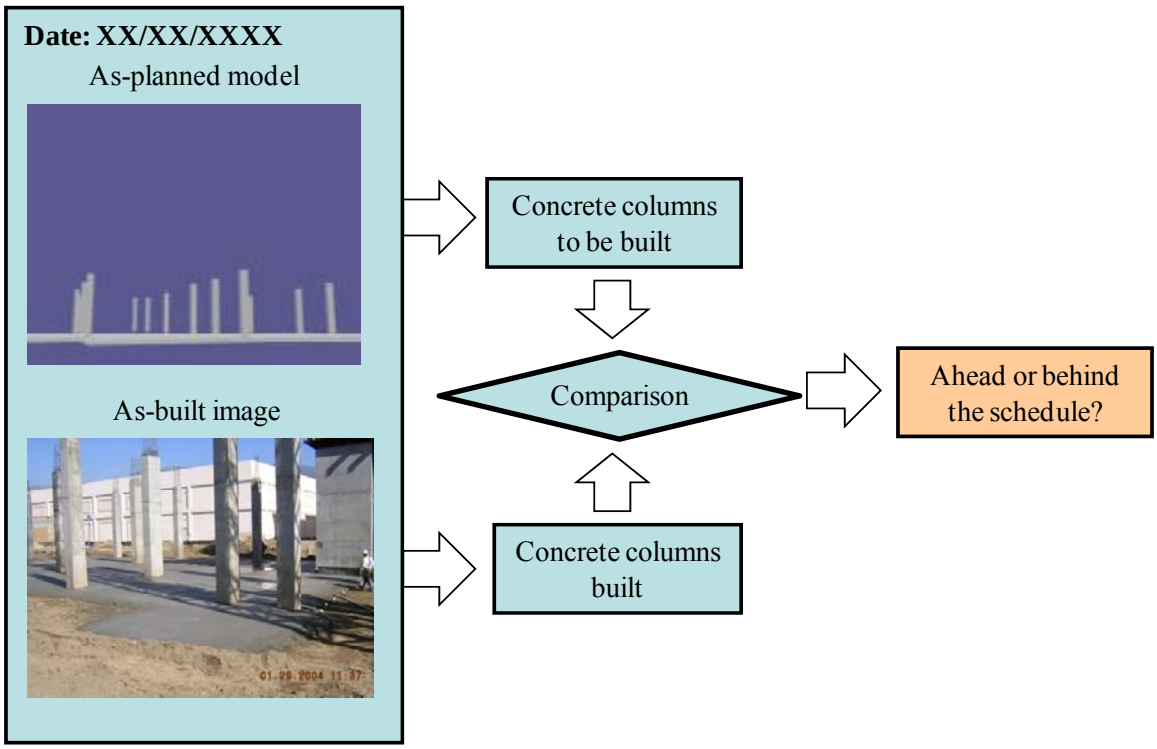


Figure 78: Progress monitoring at component level

5.6 Productivity Measurement

Construction is a trillion dollar business in the United States (ENR, 2004). Any little improvement in construction productivity may lead to huge savings in the construction industry. In order to improve construction productivity, the preliminary and important step is to measure the productivity of current construction operations. Traditional methods for productivity analysis include project level information system, direct observation, and survey/interview (Gong and Caldas, 2010). Although these methods offer plausible solutions to support productivity improvement decision making, most of them are labor- and cost- intensive in the collection of productivity information, which leads to the slow update of productivity measurement during a construction project with little details (Cheok et al. 2000).

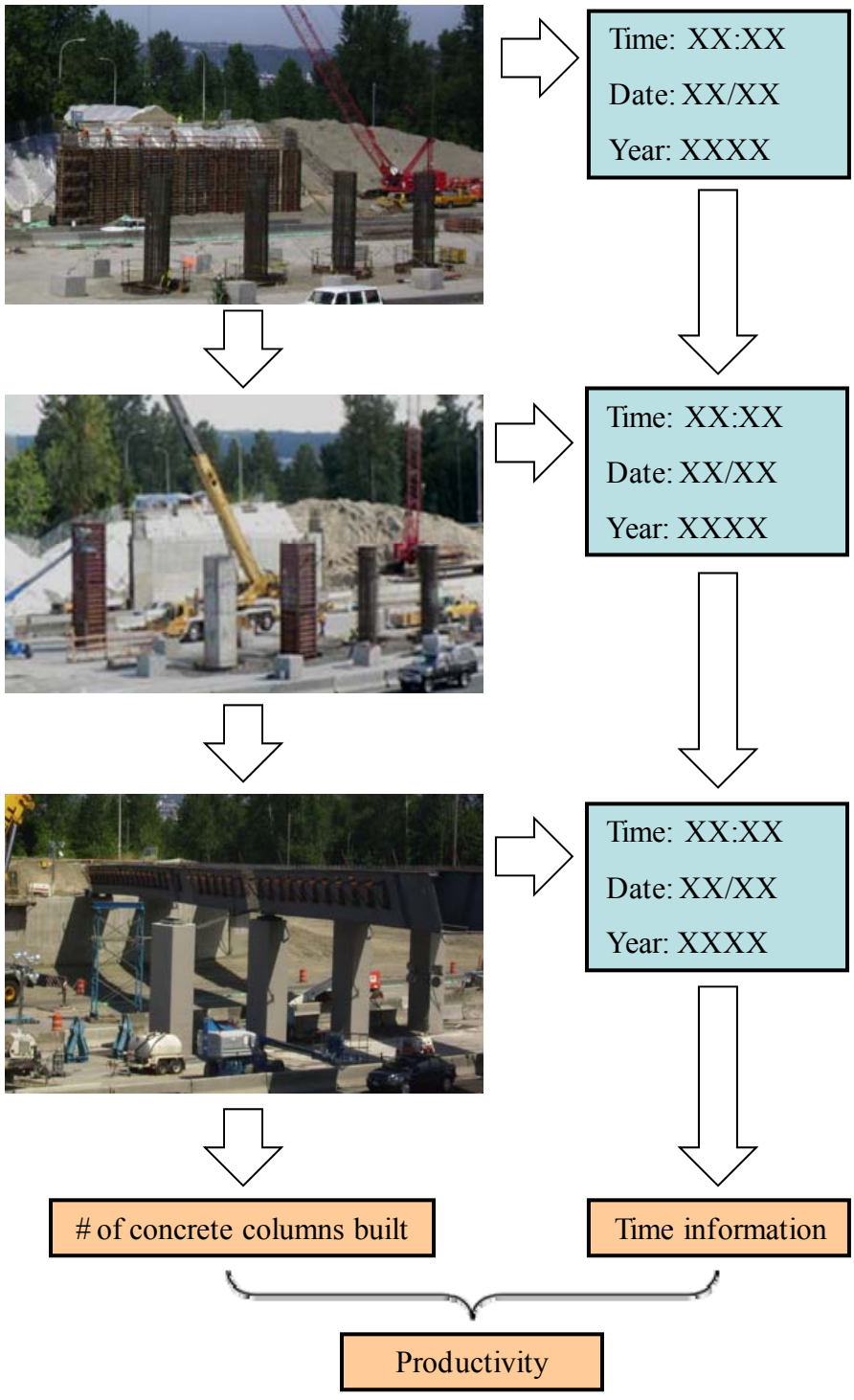


Figure 79: Productivity measurement

My research can be used to facilitate the collection of the productivity information of construction operations at a construction site. For example, it recognizes and counts the concrete columns that have been newly built from the videos captured within a certain period of time (Figure 79). This as-built element information combined with time information helps construction engineers automatically measure the productivity of pouring concrete columns at the construction site.

The research improves current data collection and analysis practices in the construction industry. It is expected to reduce the field work, time, and cost of construction engineers in collecting productivity information of construction operations at a construction site. Also, the productivity information can be collected in near real time, which lays the ground to support productivity analysis and decision making for construction engineers and project managers.

CHAPTER 6
CONCLUSIONS

This chapter first reviews the motivation and objectives of this research. Then, the brief descriptions of the methods created in this research are outlined. The conclusions, recommendations, possibilities of future research that grow out of this research are finally presented.

6.1 Review of Motivation and Objectives

Considering portions of aging and deteriorating civil infrastructure and increased risks of natural and manmade hazards and disasters, "Restoring and Improving Urban Infrastructure" has been listed as one of the Grand Challenges in the 21st century (NAE, 2008). One critical component behind this challenge lies in the difficulty of current practice in infrastructure inspection and assessment, which is not automatic and fast. For example, many advanced sensing techniques have been developed for the purpose of providing detailed and quantitative data on infrastructure conditions, but the significant amounts of time and cost associated with the techniques make them prohibitive either on a routine basis or either in an emergent scenario. Today, the first and primary step in infrastructure inspection and assessment is still manual visual inspection supplemented with other qualitative methods like judging the sounds.

Several limitations have been identified with manual visual inspection, including the subjective nature of inspection results, and time-consuming inspection process. In order to overcome these limitations, automated visual inspection methods have been proposed to enhance or replicate the current practice of manual visual inspection. These methods

are focused on distinguishing defects or damage areas from the surface background in an image with pattern recognition, digial filtering, and machine vision techniques. Their effectiveness has been validated when inspecting structures like bridges, pipes and tunnels.

Although existing automated visual inspection methods are effective in finding defects and damage, the application of these methods for automated rapid infrastructure assessment and rehabilitation is limited because of missing two important links. The first link is the connection between the defects/damage and the structural members that the defects/damage lie on. The second link is the relationship between the defects/damage and their impacts on infrastructure members. The main objective of this research is to investigate ways of how to create these two links by retrieving useful information from images with appropriate techniques in the areas of visual pattern recognition, digital filtering, and machine vision.

Consider the fact that there are different types of structures in the world, and one certain type of structures has categories of structural members with various defects/damage. It is impossible to recognize all structural members, and detect and evaluate all the defects/damage with the recognized structural members in one research study. Therefore, the research effort in this study is focused on three parts: 1) the recognition of concrete columns from images, 2) the retrieval of crack properties on concrete column surfaces, and 3) the detection of air pockets and discoloration in a concrete surface and the evaluation of their impact on the visual quality of the surface. The research work in three parts is expected to provide examples of how to create two

links for other types of structural members, defects and damage to support automated and rapid infrastructure assessment and rehabilitation in the future.

6.2 Review of Methods

Three main methods have been created in order to support the main objective of this research. The first method is concrete column recognition. A concrete column in an image is assumed to have a pair of long near-vertical lines at its sides and a concrete material region in the middle. Based on this assumption, the method first extracts long near-vertical lines from an image through edge detection and the Hough Transform. The bounding rectangle for each pair of lines is then constructed. When the rectangle resembles the shape of a column and the color and texture contained in the pair of lines indicate that they belong to concrete material, a concrete column surface is assumed to be located.

The second method created in this research study is crack properties retrieval. The method starts with the extraction of the topological skeleton points of a crack in a crack map produced by an existing crack detection method. Then, the crack is divided into different segments based on the configuration of skeleton points. In parallel to this process, the distance of crack skeleton points to the crack boundaries in the map is calculated. The distance information is combined with crack segmenting information, and the properties (e.g. width, length, and orientation) of the crack in the map can be retrieved. The retrieved properties are further related to the width of the structural element to produce relative measurements.

The third method is related to the detection of air pockets and discoloration on a concrete surface and the evaluation of their impacts on the visual quality of the surface.

In the method, a spot filter is designed and applied to a concrete surface image at different scaling levels to locate the air pockets with different size. In parallel to this process, the discoloration areas on the concrete surface are identified with image segmentation. The properties of air pockets (e.g. number and size) and discoloration (e.g. covering area) are then calculated. These properties are used to quantify the visual impacts of air pockets and discoloration on the surface visual quality. Based on the quantified visual impacts, whether the surface quality is acceptable or not can be decided.

6.3 Discussion and Conclusions

All the methods proposed in this research study have been implemented in the Microsoft Visual Studio environment. Real images and videos have been used to test these methods. The results from the methods were compared with the results retrieved manually to indicate the effectiveness of the methods. According to the test results, almost 79.1% of real concrete columns can be recognized with the method and the recognition precision can reach 89.1%. The measurement errors in relative maximum width, length, and orientation can reach 0.35%, 2.21%, and 3.29°. The precision and recall for air pockets detection can reach 91.1% and 85.6%, and most decisions automatically made based on the properties of air pockets and discoloration are matched with the decisions made by experienced inspectors. The test results retrieved from this research study indicated that the initial objectives of this research were met successfully.

In concrete column recognition, the test results indicated that concrete columns in images or videos can be recognized, when at least one of their surfaces is visible. For multiple concrete columns shown in a single image or video, the concrete columns close to a viewer can be easily recognized, while not all distant concrete columns can be

recognized, since it is difficult to extract their long near-vertical lines from the image or video. These concrete columns can be recognized when the view walks close to them.

In crack properties retrieval, the test results indicated that the crack pixels in the crack map can be grouped into different cracks correctly according to their connectivity information. In most cases, the properties retrieved from each crack are close to the manual measurement. Missing the detection of critical crack pixels connecting two crack segments leads to the error of identifying one single crack as two separate cracks. The linking operation proposed by Sinha and Fieguth (2006) may help to solve this problem, but the operation is case-dependent and empirical.

In air pockets and discoloration detection and evaluation, the test results indicated that most air pockets and discoloration areas can be located precisely on a concrete surface. The visual impact ratios for air pockets and discoloration calculated from the method can be used to objectively and accurately evaluate the quality of concrete surfaces in terms of air pockets and discoloration. Compared with the previous air pocket detection method proposed by Suwwanakarn et al. (2007), the method in this research study significantly improved the air pockets detection recall while maintaining almost same detection precision.

6.4 Contributions

This research contributed to the fundamental knowledge in retrieving the information of concrete material, concrete columns, defects and damage such as cracks, air pockets and discoloration from images. This information can be used to support automated and rapid infrastructure assessment and rehabilitation either on a routine basis or in an emergency scenario. Specifically, it can:

1. Provide quantitative inspection results without the reliance on the personal experience of each evaluator. This overcomes the subjective nature of manual visual inspection.
2. Speed up the manual inspection process. Defects and damage such as cracks, air pockets and discoloration on a concrete surface can be automatically detected. Their properties can be further retrieved for the assessment and rehabilitation purpose without many human interventions.
3. Reduce the cost associated with manual inspection. The time saved in manual evaluation will directly reduce inspection cost. Also, inspection results are not heavily dependent on the inspectors' experience anymore. Contractors or owners can use less experienced, lower cost inspectors for inspection, since only taking pictures of the under-inspection surfaces is needed during the on-site visit.
4. Alleviate the challenge of not having enough qualified inspectors by enhancing or replace the current practices of manual visual inspection.

In addition to infrastructure assessment and rehabilitation, this research is expected to influence the research efforts in the area of automation in construction, and specifically facilitate:

1. As-built infrastructure modeling. The material and structural member information retrieved from images can be used to automatically classify 3D points in the point clouds produced by photogrammetry or videogrammetry.
2. Construction progress monitoring at structural element level. The material and structural member information retrieved from as-built images can be referred to

the as-planned model of a construction project to indicate whether the project is ahead or behind the schedule.

3. Productivity measurement of construction operations at a construction site. The material and structural member information retrieved from time-lapsed images or videos of a construction operation can be used to measure the productivity of the operation.

6.5 Recommendations and Future Research

The work is part of a large research project that aims to automate and accelerate the procedure of manual visual inspection. It investigates the ways of retrieving structural element information and defects/damage information from images and videos using appropriate image processing, digital filtering, and machine vision techniques. The examples created in this research study (i.e. concrete column recognition, crack properties retrieval, and air pockets and discoloration detection and evaluation) provide valuable experience that can be used to guide the research work in the future.

First, in structural element recognition, the proposed method is limited to the recognition of concrete columns in a structure. In the future, the methods of recognizing other types of structural elements, such as beams and shear walls, can be investigated. Also, the concept developed in the method can be extended to recognize columns with other materials, such as steel or wood.

In crack properties retrieval, the method considers the merging of multiple crack segments only when they connect each other and have similar directions. How to combine the properties of two separate cracks for the perspective of structural element integrity assessment needs to be investigated in the future. One of the ultimate goals of

this research is to use the retrieved damage properties for automated structural element integrity assessment. In the assessment level, two cracks may indicate the same type of damage, even if they are separated in locations. Therefore, the properties of the cracks that show the same type of damage need to be considered together instead of separately from the assessment perspective.

In air pockets and discoloration detection and evaluation, the method assumes that the images have uniform lighting conditions, since dark shadows create large intensity variations in the images and can possibly lead to incorrect discoloration detection results. In order to improve the detection and assessment capabilities, more research is necessary. Areas of improvement can be how to remove the assumption of uniform lighting condition when detecting and evaluating air pockets and discoloration.

Overall, this research study is just a tip of the iceberg in retrieving structural element and defects/damage information from images and videos to support automated visual inspection. Much more information for civil infrastructure assessment and rehabilitation is still stored in the images and videos and waiting for us to retrieve.

APPENDIX A: CODE FOR CONCRETE COLUMN RECOGNTION

This appendix presents three main parts in the prototype code for concrete column recognition. They include near-vertical edge pixel extraction, near-vertical line detection, and ANN-based concrete material classification.

```
/*********************************************************
// Edge pixels
/*********************************************************
void LineDetector::DetectEdges()
{
	IplImage* pBlueChannel = m_pColorComponentSrcImage[nBlueChannel];
	IplImage* pGreenChannel = m_pColorComponentSrcImage[nGreenChannel];
	IplImage* pRedChannel = m_pColorComponentSrcImage[nRedChannel];

	if(pBlueChannel == NULL || pGreenChannel == NULL || pRedChannel == NULL)
		return;

	// Calculate first derivatives in x and y directions from red, green and blue channels
	cvSobel(pBlueChannel, m_pFirstDerivativeBlueX, 1, 0);
	cvSobel(pBlueChannel, m_pFirstDerivativeBlueY, 0, 1);
	cvSobel(pGreenChannel, m_pFirstDerivativeGreenX, 1, 0);
	cvSobel(pGreenChannel, m_pFirstDerivativeGreenY, 0, 1);
	cvSobel(pRedChannel, m_pFirstDerivativeRedX, 1, 0);
	cvSobel(pRedChannel, m_pFirstDerivativeRedY, 0, 1);

	float* pFirstDerivativeBlueXData = NULL;
	float* pFirstDerivativeBlueYData = NULL;
	float* pFirstDerivativeGreenXData = NULL;
	float* pFirstDerivativeGreenYData = NULL;
	float* pFirstDerivativeRedXData = NULL;
	float* pFirstDerivativeRedYData = NULL;

	cvGetRawData(m_pFirstDerivativeBlueX, (uchar**)(&pFirstDerivativeBlueXData));
	cvGetRawData(m_pFirstDerivativeBlueY, (uchar**)(&pFirstDerivativeBlueYData));
	cvGetRawData(m_pFirstDerivativeGreenX, (uchar**)(&pFirstDerivativeGreenXData));
	cvGetRawData(m_pFirstDerivativeGreenY, (uchar**)(&pFirstDerivativeGreenYData));
	cvGetRawData(m_pFirstDerivativeRedX, (uchar**)(&pFirstDerivativeRedXData));
	cvGetRawData(m_pFirstDerivativeRedY, (uchar**)(&pFirstDerivativeRedYData));


	float* pMxxData = NULL;
	float* pMxyData = NULL;
	float* pMyyData = NULL;

	int nStepofMData = 0;

	cvGetRawData(m_pMxx, (uchar**)(&pMxxData), &nStepofMData);
	cvGetRawData(m_pMxy, (uchar**)(&pMxyData));
```

```
	cvGetRawData(m_pMyy, (uchar**)(&pMyyData));

	nStepofMData /= sizeof(pMxxData[0]);

	float* pVData = NULL;
	float* pThetaData = NULL;

	cvGetRawData(m_pV, (uchar**)(&pVData));
	cvGetRawData(m_pTheta, (uchar**)(&pThetaData));

	int i, j;
	int nWidth = pBlueChannel->width;
	int nHeight = pBlueChannel->height;

	// Calculate gradient magnitude and direction of each pixel
	for( j = 0; j < nHeight; j++ )
	for( i = 0; i < nWidth; i++ )
	{
		pMxxData[j*nStepofMData + i] = pFirstDerivativeRedXData[j*nStepofMData +
i]*pFirstDerivativeRedXData[j*nStepofMData + i] +
			pFirstDerivativeGreenXData[j*nStepofMData +
i]*pFirstDerivativeGreenXData[j*nStepofMData + i] +
			pFirstDerivativeBlueXData[j*nStepofMData +
i]*pFirstDerivativeBlueXData[j*nStepofMData + i];

		pMxyData[j*nStepofMData + i] = pFirstDerivativeRedXData[j*nStepofMData +
i]*pFirstDerivativeRedYData[j*nStepofMData + i] +
			pFirstDerivativeGreenXData[j*nStepofMData +
i]*pFirstDerivativeGreenYData[j*nStepofMData + i] +
			pFirstDerivativeBlueXData[j*nStepofMData +
i]*pFirstDerivativeBlueYData[j*nStepofMData + i];

		pMyyData[j*nStepofMData + i] = pFirstDerivativeRedYData[j*nStepofMData +
i]*pFirstDerivativeRedYData[j*nStepofMData + i] +
			pFirstDerivativeGreenYData[j*nStepofMData +
i]*pFirstDerivativeGreenYData[j*nStepofMData + i] +
			pFirstDerivativeBlueYData[j*nStepofMData +
i]*pFirstDerivativeBlueYData[j*nStepofMData + i];

		pVData[j*nStepofMData + i] = ((float)sqrt(
			(pMxxData[j*nStepofMData + i] + pMyyData[j*nStepofMData +
i])*(pMxxData[j*nStepofMData + i] + pMyyData[j*nStepofMData + i]) -
			4*(pMxxData[j*nStepofMData + i]*pMyyData[j*nStepofMData + i] -
pMxyData[j*nStepofMData + i]*pMxyData[j*nStepofMData + i])) +
			pMxxData[j*nStepofMData + i] + pMyyData[j*nStepofMData + i])/2.0f;

		pThetaData[j*nStepofMData + i] = atan2(pVData[j*nStepofMData + i] -
pMxxData[j*nStepofMData + i],
			pMxyData[j*nStepofMData + i]);
	}

	CvScalar scalarMean;
	double minValue, maxValue;

	scalarMean = cvAvg(m_pV);
	cvMinMaxLoc(m_pV, &minValue, &maxValue);
```

```
        cvThreshold(m_pV, m_pV, scalarMean.val[0], maxValue, CV_THRESH_TOZERO);

        cvZero(m_pEdges);

        uchar* pEdgeData = NULL;
        int nStepofEdgeImage = 0;

        cvGetRawData(m_pEdges, &pEdgeData, &nStepofEdgeImage);

        // Find edge pixels
        float NeighborDirection[9] = { -PI, -0.75f*PI,-0.5f*PI, -0.25f*PI, 0, 0.25f*PI, 0.5f*PI, 0.75f*PI,
PI }; // Two neighbors
        int NeighborCandidateX[9] = { -1, -1,  0,  1, 1, 1, 0,-1,-1 };
        int NeighborCandidateY[9] = {  0, -1, -1, -1, 0, 1, 1, 1, 0 };


        for( j = 0; j < nHeight; j++ )
        for( i = 0; i < nWidth; i++ )
        {
                float fThetaData = pThetaData[j*nStepofMData + i];

                float fVData = pVData[j*nStepofMData + i];

                // Eight directions
                for ( int nDirection = 0; nDirection < 9; nDirection++ )
                {
                        float fAngleDiff = (float)fabs(fThetaData - NeighborDirection[nDirection]);

                        if( fAngleDiff <= 0.125f*PI)
                        {
                                int NeighborX[2], NeighborY[2];
                                float fNeighborData[2] = { -99999.0f, -99999.0f };

                                NeighborX[0] = i + NeighborCandidateX[nDirection];
                                NeighborY[0] = j + NeighborCandidateY[nDirection];

                                NeighborX[1] = i - NeighborCandidateX[nDirection];
                                NeighborY[1] = j - NeighborCandidateY[nDirection];

                                if(NeighborX[0] >= 0 && NeighborX[0] < nWidth &&
                                        NeighborY[0] >= 0 && NeighborY[0] < nHeight )
                                        fNeighborData[0] = pVData[NeighborY[0]*nStepofMData +
NeighborX[0]];

                                if(NeighborX[1] >= 0 && NeighborX[1] < nWidth &&
                                        NeighborY[1] >= 0 && NeighborY[1] < nHeight )
                                        fNeighborData[1] = pVData[NeighborY[1]*nStepofMData +
NeighborX[1]];

                                if( fVData > fNeighborData[0] && fVData > fNeighborData[1] )
                                        pEdgeData[j*nStepofEdgeImage + i] = chEdgeIntensity;
                        }
                }
        }

   for( j = 0; j < nHeight; j++ )
```

```
	for( i = 0; i < nWidth; i++ )
	{
		uchar chEdgeData = pEdgeData[j*nStepofEdgeImage + i];

		if( chEdgeData == 0) continue;

		int nNeighborEdge = 0;
		float fSumofNeighborEdgeValue = 0;
		float fSumofDiffbtwNeighborEdgeValue = 0;

		for( int nDirection = 0; nDirection < 8; nDirection++ )
		{
			int nNeighborX = i + NeighborCandidateX[nDirection];
			int nNeighborY = j + NeighborCandidateY[nDirection];

			if(nNeighborX >= 0 && nNeighborX < nWidth &&
				nNeighborY >= 0 && nNeighborY < nHeight )
			{
				if( pEdgeData[nNeighborY*nStepofEdgeImage + nNeighborX] ==
chEdgeIntensity )
				{
					nNeighborEdge = 1;

					fSumofNeighborEdgeValue +=
pVData[nNeighborY*nStepofMData + nNeighborX];
					fSumofDiffbtwNeighborEdgeValue += (float)fabs(
						pVData[j*nStepofMData + i] -
						pVData[nNeighborY*nStepofMData + nNeighborX]);
				}
			}
		}

		if( fSumofDiffbtwNeighborEdgeValue - fSumofNeighborEdgeValue > 0 ||
			nNeighborEdge == 0 )
			pEdgeData[j*nStepofEdgeImage + i] = 0;
	}
}

/*********************************************************
// Near-vertical edge pixels
*********************************************************/
void LineDetector::DetectVertEdges(float fVerticalTheta)
{
	cvZero(m_pVertEdges);

	// Theta data
	float* pThetaData = NULL;
	int nStepofTheta;
	cvGetRawData(m_pTheta, (uchar**)(&pThetaData), &nStepofTheta);
	nStepofTheta /= sizeof(pThetaData[0]);

	// Edge data
	uchar* pEdgeData = NULL;
	CvSize szImage;
	int nStepofEdgeImage = 0;
	cvGetRawData(m_pEdges, &pEdgeData, &nStepofEdgeImage, &szImage);
```

```
	nStepofEdgeImage /= sizeof(pEdgeData[0]);

	// Edge data with horizontal orientation information
	uchar* pVertEdgeData = NULL;
	cvGetRawData(m_pVertEdges, &pVertEdgeData);

	int i = 0, j = 0;
	for( j = 0; j < szImage.height; j++ )
	for( i = 0; i < szImage.width; i++ )
	{
		uchar chEdgeData = pEdgeData[j*nStepofEdgeImage + i];
		if(chEdgeData == 0) continue;

		float fThetaData = pThetaData[j*nStepofTheta + i];

		if ( fabs(fThetaData) < fVerticalTheta ||
			fabs(fThetaData - PI) < fVerticalTheta ||
			fabs(fThetaData + PI) < fVerticalTheta )
			pVertEdgeData[j*nStepofEdgeImage + i] = 255;
	}

//	cvCvtColor(m_pVertEdges , m_pEdgeImage, CV_GRAY2BGR);
}

/*********************************************************
// Near-vertical vertical lines – Hough Transform
// The code was modified from the Code in OpenCV (http://opencv.willowgarage.com/wiki/)
*********************************************************/
void LineDetector::DetectVertLineSegments(int nDeltaTheta, int nDeltaRho, int nMinAcc, int nMinLength,
int nMaxGap)
{

	// Create memory storage that will contain all the dynamic data
	CvMemStorage* pLineMemStorage = NULL;
	pLineMemStorage = cvCreateMemStorage(0);

	CvSeq* pVertLines = NULL;
	pVertLines = HoughLineTransform( m_pVertEdges, pLineMemStorage,
		(float)nDeltaRho, nDeltaTheta/180.0f, nMinAcc, nMinLength, nMaxGap);

	m_vecLineSegments.clear();
	int nLines = 0;
	for ( nLines = 0; nLines < pVertLines->total; nLines++ )
	{
		CvPoint* pPoints = (CvPoint*)cvGetSeqElem(pVertLines,nLines);

		LineSegment lnSegment;
		lnSegment.m_ptStart = pPoints[0].y <= pPoints[1].y ? pPoints[0]: pPoints[1];
		lnSegment.m_ptEnd = pPoints[0].y > pPoints[1].y ? pPoints[0]: pPoints[1];
		lnSegment.m_fLength = CalculateDistance(pPoints[0], pPoints[1]);
		m_vecLineSegments.push_back(lnSegment);
	}
}

CvSeq* LineDetector:: HoughLineTransform( CvArr* src_image, void* lineStorage, float rho, float theta,
int threshold,
```

```
            int param1, int param2 )
{
            if ( lineStorage == NULL || ( rho <= 0 || theta <= 0 || threshold <= 0 ) ) return NULL;

            CvMat stub, *img = (CvMat*)src_image;
            img = cvGetMat( img, &stub );

            int lineType = CV_32SC4, elemSize = sizeof(int)*4;
            int linesMax = INT_MAX;
            int iparam1 = cvRound(param1), iparam2 = cvRound(param2);

            CvSeq* lines = cvCreateSeq( lineType, sizeof(CvSeq), elemSize, (CvMemStorage*)lineStorage );

            HoughLinesProbabalistic( img, rho, theta, threshold,
                        param1, param2, lines, linesMax );

            return lines;
}

void LineDetector:: HoughLinesProbabalistic( CvMat* image, float rho, float theta, int threshold,
            int lineLength, int lineGap, CvSeq *lines, int linesMax )
{
            // Theta data
            float* pThetaData = NULL;
            int nStepofTheta;
            cvGetRawData(m_pTheta, (uchar**)(&pThetaData), &nStepofTheta);
            nStepofTheta /= sizeof(pThetaData[0]);

            CvMat* accum = 0;
            CvMat* mask = 0;
            CvMat* trigtab = 0;
            CvMemStorage* storage = 0;

            CvSeq* seq;
            CvSeqWriter writer;
            int width, height;
            int numangle, numrho;
            float ang;
            int r, n, count;
            CvPoint pt;
            float irho = 1 / rho;
            CvRNG rng = cvRNG(-1);
            const float* ttab;
            uchar* mdata0;

            width = image->cols;
            height = image->rows;

            numangle = cvRound(CV_PI / theta) + 1;
            numrho = cvRound(((width + height) * 2 + 1) / rho);

            accum = cvCreateMat( numangle, numrho, CV_32SC1 );
            mask = cvCreateMat( height, width, CV_8UC1 );
            trigtab = cvCreateMat( 1, numangle, CV_32FC2 );
            cvZero( accum );
```

```
storage = cvCreateMemStorage(0);

for( ang = 0, n = 0; n < numangle; ang += theta, n++ )
{
        trigtab->data.fl[n*2] = (float)(cos(ang) * irho);
        trigtab->data.fl[n*2+1] = (float)(sin(ang) * irho);
}
ttab = trigtab->data.fl;
mdata0 = mask->data.ptr;

cvStartWriteSeq( CV_32SC2, sizeof(CvSeq), sizeof(CvPoint), storage, &writer );

// stage 1. collect non-zero image points
for( pt.y = 0, count = 0; pt.y < height; pt.y++ )
{
        const uchar* data = image->data.ptr + pt.y*image->step;
        uchar* mdata = mdata0 + pt.y*width;
        for( pt.x = 0; pt.x < width; pt.x++ )
        {
                if( data[pt.x] )
                {
                        mdata[pt.x] = (uchar)1;
                        CV_WRITE_SEQ_ELEM( pt, writer );
                }
                else
                        mdata[pt.x] = 0;
        }
}

seq = cvEndWriteSeq( &writer );
count = seq->total;

// stage 2. process all the points in random order
for( ; count > 0; count-- )
{
        // choose random point out of the remaining ones
        int idx = cvRandInt(&rng) % count;
        int max_val = threshold-1, max_n = 0;
        CvPoint* pt = (CvPoint*)cvGetSeqElem( seq, idx );
        CvPoint line_end[2] = {{0,0}, {0,0}};
        float a, b;
        int* adata = accum->data.i;
        int i, j, k, x0, y0, dx0, dy0, xflag;
        int good_line;
        const int shift = 16;

        i = pt->y;
        j = pt->x;

        // "remove" it by overriding it with the last element
        *pt = *(CvPoint*)cvGetSeqElem( seq, count-1 );

        // check if it has been excluded already (i.e. belongs to some other line)
        if( !mdata0[i*width + j] )
                continue;
```

```
                    // update accumulator, find the most probable line
//                          for( n = 0; n < numangle; n++, adata += numrho )
                    float fThetaData =  pThetaData[i*nStepofTheta+j];
                    if( fThetaData < 0 ) fThetaData = fThetaData + PI;
//                          if( PI - fThetaData < theta ) fThetaData = PI - fThetaData;

                    n = cvRound( fThetaData / theta);
                    {
                              r = cvRound( j * ttab[n*2] + i * ttab[n*2+1] );
                              r += (numrho - 1) / 2;
                              int val = ++adata[n*numrho+r];
                              if( max_val < val )
                              {
                                        max_val = val;
                                        max_n = n;
                              }
                    }

                    // if it is too "weak" candidate, continue with another point
                    if( max_val < threshold )
                              continue;

                    // from the current point walk in each direction
                    // along the found line and extract the line segment
                    a = -ttab[max_n*2+1];
                    b = ttab[max_n*2];
                    x0 = j;
                    y0 = i;
                    if( fabs(a) > fabs(b) )
                    {
                              xflag = 1;
                              dx0 = a > 0 ? 1 : -1;
                              dy0 = cvRound( b*(1 << shift)/fabs(a) );
                              y0 = (y0 << shift) + (1 << (shift-1));
                    }
                    else
                    {
                              xflag = 0;
                              dy0 = b > 0 ? 1 : -1;
                              dx0 = cvRound( a*(1 << shift)/fabs(b) );
                              x0 = (x0 << shift) + (1 << (shift-1));
                    }

                    for( k = 0; k < 2; k++ )
                    {
                              int gap = 0, x = x0, y = y0, dx = dx0, dy = dy0;

                              if( k > 0 )
                                        dx = -dx, dy = -dy;

                              // walk along the line using fixed-point arithmetics,
                              // stop at the image border or in case of too big gap
                              for( ;; x += dx, y += dy )
                              {
                                        uchar* mdata;
                                        int i1, j1;
```

```
                    if( xflag )
                    {
                            j1 = x;
                            i1 = y >> shift;
                    }
                    else
                    {
                            j1 = x >> shift;
                            i1 = y;
                    }

                    if( j1 < 0 || j1 >= width || i1 < 0 || i1 >= height )
                            break;

                    mdata = mdata0 + i1*width + j1;


                    if( *mdata )
                    {
                            gap = 0;
                            line_end[k].y = i1;
                            line_end[k].x = j1;
                    }
                    else if( ++gap > lineGap )
                            break;
            }
    }

    good_line = abs(line_end[1].x - line_end[0].x) >= lineLength ||
                            abs(line_end[1].y - line_end[0].y) >= lineLength;

    for( k = 0; k < 2; k++ )
    {
            int x = x0, y = y0, dx = dx0, dy = dy0;

            if( k > 0 )
                    dx = -dx, dy = -dy;

            // walk along the line using fixed-point arithmetics,
            // stop at the image border or in case of too big gap
            for( ;; x += dx, y += dy )
            {
                    uchar* mdata;
                    int i1, j1;

                    if( xflag )
                    {
                            j1 = x;
                            i1 = y >> shift;
                    }
                    else
                    {
                            j1 = x >> shift;
                            i1 = y;
                    }
```

```
                                        mdata = mdata0 + i1*width + j1;


                                        if( *mdata )
                                        {
                                                if( good_line )
                                                {
                                                        adata = accum->data.i;
                                                        for( n = 0; n < numangle; n++, adata += numrho )
                                                        {
                                                                r = cvRound( j1 * ttab[n*2] + i1 *
ttab[n*2+1] );
                                                                r += (numrho - 1) / 2;
                                                                adata[r]--;
                                                        }
                                                }
                                                *mdata = 0;
                                        }

                                        if( i1 == line_end[k].y && j1 == line_end[k].x )
                                                break;
                                }
                        }

                        if( good_line )
                        {
                                CvRect lr = { line_end[0].x, line_end[0].y, line_end[1].x, line_end[1].y };
                                cvSeqPush( lines, &lr );
                                if( lines->total >= linesMax )
                                        exit(-1);  // EXIT;
                        }
                }

                cvReleaseMat( &accum );
                cvReleaseMat( &mask );
                cvReleaseMat( &trigtab );
                cvReleaseMemStorage( &storage );

}

/*********************************************************
// ANN-based concrete material classificaiton
*********************************************************/
struct fann_train_data* fann_read_train_from_mem(float* input_data, float* output_data,
                                        int num_data, int num_input, int num_output )
{
  struct fann_train_data* data = (struct fann_train_data *) malloc(sizeof(struct fann_train_data));

  fann_type* data_input;
  fann_type* data_output;

  int i, j;

        if(data == NULL)
        {
```

```
		fann_error(NULL, FANN_E_CANT_ALLOCATE_MEM);
		return NULL;
	}

	fann_init_error_data((struct fann_error *) data);

	data->num_data = num_data;
	data->num_input = num_input;
	data->num_output = num_output;
data->input = (fann_type **) calloc(num_data, sizeof(fann_type *));

	if(data->input == NULL)
	{
		fann_error(NULL, FANN_E_CANT_ALLOCATE_MEM);
		fann_destroy_train(data);
		return NULL;
	}

	data->output = (fann_type **) calloc(num_data, sizeof(fann_type *));
	if(data->output == NULL)
	{
		fann_error(NULL, FANN_E_CANT_ALLOCATE_MEM);
		fann_destroy_train(data);
		return NULL;
	}

	data_input = (fann_type *) calloc(num_input * num_data, sizeof(fann_type));
	if(data_input == NULL)
	{
		fann_error(NULL, FANN_E_CANT_ALLOCATE_MEM);
		fann_destroy_train(data);
		return NULL;
	}

	data_output = (fann_type *) calloc(num_output * num_data, sizeof(fann_type));
	if(data_output == NULL)
	{
		fann_error(NULL, FANN_E_CANT_ALLOCATE_MEM);
		fann_destroy_train(data);
		return NULL;
	}

	for(i = 0; i != num_data; i++)
	{
		data->input[i] = data_input;
		data_input += num_input;

		for(j = 0; j != num_input; j++)
			data->input[i][j] = input_data[i*num_input+j];

		data->output[i] = data_output;
		data_output += num_output;

		for(j = 0; j != num_output; j++)
			data->output[i][j] = output_data[i*num_output+j];
	}
```

```
        return data;
}

int RegionClassifier:: Classify(MaterialSignature signature)
{
        float input_data[] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f};
        float output_data[] = {1.0f};

                input_data[0] = signature.r2g_mean;
                input_data[1] = signature.r2g_std;
                input_data[2] = signature.b2g_mean;
                input_data[3] = signature.b2g_std;
                input_data[4] = signature.edge_max;
                input_data[5] = signature.bar_max;

                output_data[0] = 0.0f;
                data = fann_read_train_from_mem(input_data, output_data, 1, 6, 1);
                calc_out = fann_test(m_pANN, data->input[0], data->output[0]);
                if( calc_out[0] > 0.0f )
                        return 1; // Concrete
                return -1 // Non-concrete;
                                                }
```

APPENDIX B: CODE FOR CRACK PROPERTIES RETREIVAL

This appendix presents the main part in the prototype code for crack properties retrieval. It includes crack skeleton extraction, distance transform, crack segmentation, and crack merge.

```
/*********************************************************
// retrieve the properties of cracks with a provided crack map (i.e. m_pCrackMap)
*********************************************************/
void CCrackDetection::RetrieveCrackProperties()
{
	if(m_pCrackMap == NULL) return;

	int	nMinCrackSegmentSize = 10;
	float	fMaxOrienDifference = 45.0f;

	DetectCrackSkeleton();
	DetectCrackDistance();
	DetectCrackSegments();

	ReduceCrackSegments(nMinCrackSegmentSize, m_vecCracks, m_vecSimplifiedCracks);
	CalculateCrackSegmentOrientation(m_vecSimplifiedCracks);
	MergeCrackSegments(fMaxOrienDifference, m_vecSimplifiedCracks);
	CalculateCrackProperties(m_vecSimplifiedCracks);
}

void CCrackDetection::CalculateCrackSegmentOrientation(vector<Crack>& vecCracks)
{
	CvMemStorage* pointStorage = cvCreateMemStorage();

	int nCrack = 0;
	int nCrackSegment = 0;
	int nCrackPoint = 0;

	for( nCrack = 0; nCrack < vecCracks.size(); nCrack++ )
	{
		for( nCrackSegment = 0;
		     nCrackSegment < vecCracks[nCrack].m_vecCrackSegments.size();
		     nCrackSegment++ )
		{
			CvSeq *points =
 cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq), sizeof(CvPoint), pointStorage );

			for( nCrackPoint = 0;
			     nCrackPoint <
vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints.size();
			     nCrackPoint++ )
			{
```

```
                                        CrackPoint crack_point =
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints[nCrackPoint];
                                        CvPoint pt;
                                        pt.x = crack_point.m_nX; pt.y = crack_point.m_nY;
                                        cvSeqPush(points, &pt);
                                }
                                CvBox2D box = cvMinAreaRect2(points, pointStorage);
                                if(box.size.height >= box.size.width)
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D = box;
                                else
                                {
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D.center = box.center;
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D.size.width =
box.size.height;
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D.size.height =
box.size.width;
        vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D.angle = box.angle + 90;
                                }
                        }
                }
                cvReleaseMemStorage(&pointStorage);
        }
        void CCrackDetection:: CalculateCrackProperties(const vector<Crack>& vecCracks)
        {
                m_vecCrackProperties.clear();
                if(m_pCrackSkeleton == NULL || m_pCrackDistance == NULL) return;
                CvSize szImage;
                uchar* pSkeletonData = NULL;
                int nStepofSkeleton = 0;
                cvGetRawData(m_pCrackSkeleton, &pSkeletonData, &nStepofSkeleton, &szImage);
                nStepofSkeleton /= sizeof(pSkeletonData[0]);
                float* pDistanceData = NULL;
                int nStepofDistance = 0;
                cvGetRawData(m_pCrackDistance, (uchar**)&pDistanceData, &nStepofDistance);
                nStepofDistance /= sizeof(pDistanceData[0]);
                int nCrack, nSegment, nPoint;
                int nCrackSize = vecCracks.size();
```

```
CrackProperty newCrackProperty;
newCrackProperty.m_vecSegProperties.reserve(100);

for( nCrack = 0; nCrack < nCrackSize; nCrack++ )
{
	newCrackProperty.m_vecSegProperties.clear();

	for ( nSegment = 0; nSegment <
vecCracks[nCrack].m_vecCrackSegments.size(); nSegment++ )
	{
		float maxSegmentWidth = 0;
		float avgSegmentWidth = 0;

		for( nPoint = 0; nPoint <
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.size(); nPoint++ )
		{
			CrackPoint crack_point =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints[nPoint];

			int nX = crack_point.m_nX;
			int nY = crack_point.m_nY;

			if( nX < 0 || nX > szImage.width ||
				nY < 0 || nY > szImage.height )
				continue;

			float skeletonPointWidth =
2*pDistanceData[(nY)*nStepofDistance + (nX)];

			if( maxSegmentWidth < skeletonPointWidth )
			{
				maxSegmentWidth = skeletonPointWidth;
			}
			avgSegmentWidth += skeletonPointWidth;
		}

		avgSegmentWidth /= nPoint;

		CvBox2D Obb2D =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_cvOBB2D;

		float cosTheta = cos((Obb2D.angle - 90)*PI/180);
		float sinTheta = sin((Obb2D.angle - 90)*PI/180);

		CvPoint2D32f new_pt;
		CvPoint2D32f pt[4];

		new_pt.x = ((Obb2D.size.width)/2);
		new_pt.y = ((Obb2D.size.height)/2);

		pt[0].x = (cosTheta*new_pt.x + sinTheta*new_pt.y + Obb2D.center.x);
		pt[0].y = (-sinTheta*new_pt.x + cosTheta*new_pt.y + Obb2D.center.y);

		new_pt.x = ((Obb2D.size.width)/2);
		new_pt.y = (-(Obb2D.size.height)/2);
```

```
                                        pt[1].x = (cosTheta*new_pt.x + sinTheta*new_pt.y + Obb2D.center.x);
                                        pt[1].y = (- sinTheta*new_pt.x + cosTheta*new_pt.y +
Obb2D.center.y);


                                        new_pt.x = (-(Obb2D.size.width)/2);
                                        new_pt.y = (-(Obb2D.size.height)/2);

                                        pt[2].x = (cosTheta*new_pt.x + sinTheta*new_pt.y + Obb2D.center.x);
                                        pt[2].y = (- sinTheta*new_pt.x + cosTheta*new_pt.y +
Obb2D.center.y);


                                        new_pt.x = (-(Obb2D.size.width)/2);
                                        new_pt.y = ((Obb2D.size.height)/2);

                                        pt[3].x = (cosTheta*new_pt.x + sinTheta*new_pt.y + Obb2D.center.x);
                                        pt[3].y = (- sinTheta*new_pt.x + cosTheta*new_pt.y +
Obb2D.center.y);


                                        SegmentProperty segProperty;

                                        // Absolute properties
                                        segProperty.m_fAvgWidth = avgSegmentWidth;
                                        segProperty.m_fMaxWidth = maxSegmentWidth;
                                        segProperty.m_fLength = Obb2D.size.height;
                                        segProperty.m_fOrientation = Obb2D.angle;
                                        segProperty.m_fAvgWidth2ColumnWidth =
avgSegmentWidth/szImage.width;
                                        segProperty.m_fMaxWidth2ColumnWidth =
maxSegmentWidth/szImage.width;
                                        segProperty.m_fLength2ColumnWidth =
Obb2D.size.height/szImage.width;

                                        segProperty.m_ptLocations[0] =
                                                cvPoint(cvRound((pt[0].x + pt[3].x)*0.5), cvRound((pt[0].y +
pt[3].y)*0.5));
                                        segProperty.m_ptLocations[1] =
                                                cvPoint(cvRound((pt[1].x + pt[2].x)*0.5), cvRound((pt[1].y +
pt[2].y)*0.5));


                                        newCrackProperty.m_vecSegProperties.push_back(segProperty);
                                }

                                if( newCrackProperty.m_vecSegProperties.size() > 0 )
                                        m_vecCrackProperties.push_back(newCrackProperty);
                        }

                }


        void CCrackDetection:: MergeCrackPoints(int nCrack, vector<Crack>& vecCracks )
                {
                        IplImage* pSegmentMap = cvCloneImage(m_pCrackSkeleton);
```

```
                cvConvertScaleAbs(pSegmentMap, pSegmentMap, 1.0f/255, 0);

                int NeighborCandidateX[8] = {-1, -1, -1,  0,  1, 1, 1, 0};
                int NeighborCandidateY[8] = { 1,  0, -1, -1, -1, 0, 1, 1};

                CvSize szImage;
                int    nStep = 0;
                uchar* pMapData = NULL;
                cvGetRawData(pSegmentMap, &pMapData, &nStep, &szImage);
                nStep = nStep/sizeof(pMapData[0]);

                int nSegment = 0;
                for( nSegment = 0; nSegment < vecCracks[nCrack].m_vecCrackSegments.size();
nSegment++ )
                {
                        for( int nPoint = 0; nPoint <

        vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.size(); nPoint++ )
                        {
                                CrackPoint pt =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints[nPoint];

                                assert(pMapData[pt.m_nY*nStep + pt.m_nX] == 1 );

                                pMapData[pt.m_nY*nStep + pt.m_nX] = 2;

                        }
                }

                vector<CrackPoint> vecNeighbors;
                vecNeighbors.reserve(50);
                vecNeighbors.clear();

                for( nSegment = 0; nSegment < vecCracks[nCrack].m_vecCrackSegments.size();
nSegment++ )
                {
                        int nPointSize =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.size();
                        CrackPoint stPoint =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints[0];
                        CrackPoint endPoint =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints[nPointSize - 1];

                        FindNeighboringPoints(stPoint,  pSegmentMap, vecNeighbors);
                        vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.insert(

        vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.end(),
                                vecNeighbors.begin(),
                                vecNeighbors.end());

                        FindNeighboringPoints(endPoint, pSegmentMap, vecNeighbors);
                        vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.insert(

        vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.end(),
                                vecNeighbors.begin(),
                                vecNeighbors.end());
```

```
		}

		SAFE_RELEASE_IMAGE(pSegmentMap);
	}

	void CCrackDetection:: MergeCrackSegments(float fMaxThreshOrientation, vector<Crack>&
vecCracks)
	{
		CvMemStorage* pointStorage = cvCreateMemStorage();

		IplImage* pSegmentMap = cvCloneImage(m_pCrackSkeleton);
		cvZero(pSegmentMap);

		vector<Crack> vecTempCracks = vecCracks;
		vecCracks.clear();

		int nCrack;
		int nSegment;

		Crack		newCrack;
		newCrack.m_vecCrackSegments.reserve(100);
		newCrack.m_vecCrackSegments.clear();

		CrackSegment	newSegment;
		newSegment.m_vecCrackPoints.reserve(100);
		newSegment.m_vecCrackPoints.clear();

		int		nPrevSegment = -1, nNextSegment = -1;
		stack<int>	stkSegmentNumbers;

		for( nCrack = 0; nCrack < vecTempCracks.size(); nCrack++ )
		{
			newCrack.m_vecCrackSegments.clear();

			int nSegmentSize = vecTempCracks[nCrack].m_vecCrackSegments.size();
			int* pSegmentMerged = new int[nSegmentSize];
			memset(pSegmentMerged, 0, sizeof(int)*nSegmentSize);

			InitCrackSegmentMap(vecTempCracks, nCrack, pSegmentMap);

			for( nSegment = 0; nSegment <
vecTempCracks[nCrack].m_vecCrackSegments.size(); nSegment++ )
			{
				if( pSegmentMerged[nSegment] != 0 ) continue;

				float seedOrien =
vecTempCracks[nCrack].m_vecCrackSegments[nSegment].m_cvOBB2D.angle;

				newSegment.m_vecCrackPoints.clear();

				stkSegmentNumbers.push(nSegment);
				pSegmentMerged[nSegment] = 1;

				while ( !stkSegmentNumbers.empty() )
				{
					int nCrackSegment = stkSegmentNumbers.top();
```

```
                                        stkSegmentNumbers.pop();

        newSegment.m_vecCrackPoints.insert(newSegment.m_vecCrackPoints.end(),
        vecTempCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints.begin(),
        vecTempCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints.end() );

                                        vector<int> vecNeighbors;
                                        vecNeighbors.reserve(10);
                                        vecNeighbors.clear();

                                        FindNeighboringSegments( vecTempCracks, nCrack,
nCrackSegment, pSegmentMap, vecNeighbors);

                                        float candOrien, fDegreeDiff;

                                        for( int nNeighbor = 0; nNeighbor < vecNeighbors.size();
nNeighbor++ )
                                        {
                                                int nNextSegment = vecNeighbors[nNeighbor];

                                                if ( nNextSegment >=0 && nNextSegment <
m_vecCracks[nCrack].m_vecCrackSegments.size() &&
                                                        pSegmentMerged[nNextSegment] == 0 )
                                                {

                                                        candOrien =
vecTempCracks[nCrack].m_vecCrackSegments[nNextSegment].m_cvOBB2D.angle;
                                                        fDegreeDiff = fabs(candOrien - seedOrien);

                                                        if( ( fDegreeDiff < fMaxThreshOrientation ||
180 - fDegreeDiff < fMaxThreshOrientation ) )
                                                        {

        stkSegmentNumbers.push(nNextSegment);
                                                                pSegmentMerged[nNextSegment]
= 1;
                                                        }

                                                }
                                        }

                                }

                                UpdateSegmentOBB(newSegment, pointStorage);
                                newCrack.m_vecCrackSegments.push_back(newSegment);
                        }

                        vecCracks.push_back(newCrack);

                        SortCrackSegmentbySize(vecCracks, vecCracks.size()-1);

                        delete[] pSegmentMerged; pSegmentMerged = NULL;
                }
```

```
		SAFE_RELEASE_IMAGE(pSegmentMap);
		cvReleaseMemStorage(&pointStorage); pointStorage = NULL;

	}

	void CCrackDetection:: FindNeighborCrackSegments( const vector<Crack>& vecCracks,
		int nCrack, int nCrackSegment, float fMaxThreshOrientation, float fSeedOrien, int&
nPrevSegment, int& nNextSegment)
	{
		nPrevSegment = nNextSegment = -1;

		if ( nCrack < 0 || nCrack >= vecCracks.size() ) return;
		if ( nCrackSegment < 0 || nCrackSegment >=
vecCracks[nCrack].m_vecCrackSegments.size() ) return;

		CvPoint2D32f seedPoint =
vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_cvOBB2D.center;

		float seedOrien = fSeedOrien;

		int nSegment;
		float fPrevDist = 99999, fNextDist = 99999;

		for ( nSegment = 0; nSegment < vecCracks[nCrack].m_vecCrackSegments.size();
nSegment++ )
		{
			if ( nSegment == nCrackSegment ) continue;

			CvPoint2D32f candPoint =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_cvOBB2D.center;

			float	fDist = EculideanDistance(seedPoint, candPoint);

			{
				if( seedOrien < 45 || seedOrien > 135 )
				{
					if ( candPoint.x - seedPoint.x > 0 && fDist - fNextDist < 0 )
{ fNextDist = fDist; nNextSegment = nSegment; }

					if ( candPoint.x - seedPoint.x <= 0 && fDist - fPrevDist < 0 )
{ fPrevDist = fDist; nPrevSegment = nSegment; }
				}

				else
				{
					if ( candPoint.y - seedPoint.y > 0 && fDist - fNextDist < 0 )
{ fNextDist = fDist; nNextSegment = nSegment; }

					if ( candPoint.y - seedPoint.y <= 0 && fDist - fPrevDist < 0 )
{ fPrevDist = fDist; nPrevSegment = nSegment; }

				}
			}
		}
	}
```

```
	void CCrackDetection:: FindNeighboringSegments( const vector<Crack>& vecCracks, int nCrack,
int nCrackSegment, IplImage* pMap, vector<int>& vecNeighbors)
	{
		vecNeighbors.clear();

		if ( nCrack < 0 || nCrack >= vecCracks.size() ) return;
		if ( nCrackSegment < 0 || nCrackSegment >=
vecCracks[nCrack].m_vecCrackSegments.size() ) return;

		int nDirection = 0;
		int NeighborCandidateX[8] = {-1, -1, -1,  0,  1, 1, 1, 0};
		int NeighborCandidateY[8] = { 1,  0, -1, -1, -1, 0, 1, 1};

		CvSize szImage;
		int    nStep = 0;
		uchar* pMapData = NULL;
		cvGetRawData(pMap, &pMapData, &nStep, &szImage);
		nStep = nStep/sizeof(pMapData[0]);

		for ( int nPoint = 0; nPoint <
vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints.size(); nPoint++ )
		{
			CrackPoint seedPoint =
vecCracks[nCrack].m_vecCrackSegments[nCrackSegment].m_vecCrackPoints[nPoint];

			for( nDirection = 0; nDirection < 8; nDirection++ )
			{
				CrackPoint neighborPoint;

				neighborPoint.m_nX = seedPoint.m_nX +
NeighborCandidateX[nDirection];
				neighborPoint.m_nY = seedPoint.m_nY +
NeighborCandidateY[nDirection];

				if( neighborPoint.m_nX < 0 || neighborPoint.m_nX >= szImage.width ||
					neighborPoint.m_nY < 0 || neighborPoint.m_nY >=
szImage.height )
						continue;

				int nSegment = pMapData[neighborPoint.m_nY*nStep +
neighborPoint.m_nX];
				if( nSegment > 0 && nSegment != nCrackSegment )
				{
					vecNeighbors.push_back(nSegment);
				}
			}
		}
	}

	void CCrackDetection:: FindNeighboringPoints (const CrackPoint startPoint, IplImage* pMap,
vector<CrackPoint>& vecPoints)
	{
		vecPoints.clear();
```

```
		int nDirection = 0;
		int NeighborCandidateX[8] = {-1, -1, -1,  0,  1, 1, 1, 0};
		int NeighborCandidateY[8] = { 1,  0, -1, -1, -1, 0, 1, 1};

		CvSize szImage;
		int    nStep = 0;
		uchar* pMapData = NULL;
		cvGetRawData(pMap, &pMapData, &nStep, &szImage);
		nStep = nStep/sizeof(pMapData[0]);

		stack<CrackPoint> stkPoints;
		stkPoints.push(startPoint);

		while( stkPoints.empty() == false )
		{
			CrackPoint seedPoint = stkPoints.top();
			stkPoints.pop();

			for( nDirection = 0; nDirection < 8; nDirection++ )
			{
				CrackPoint neighborPoint;

				neighborPoint.m_nX = seedPoint.m_nX +
NeighborCandidateX[nDirection];
				neighborPoint.m_nY = seedPoint.m_nY +
NeighborCandidateY[nDirection];

				if( neighborPoint.m_nX < 0 || neighborPoint.m_nX >= szImage.width ||
					neighborPoint.m_nY < 0 || neighborPoint.m_nY >=
szImage.height )
						continue;

			  if( pMapData[neighborPoint.m_nY*nStep + neighborPoint.m_nX] == 1 )
				{

					vecPoints.push_back(neighborPoint);
					stkPoints.push(neighborPoint);

					pMapData[neighborPoint.m_nY*nStep +
neighborPoint.m_nX] = 2;
				}
			}
		}
	}


	void CCrackDetection:: InitCrackSegmentMap(const vector<Crack>& vecCracks, const int
nCrack, IplImage* pMap)
	{
		cvZero(pMap);

		if( nCrack < 0 || nCrack >= vecCracks.size() ) return;

		CvSize szImage;
		int    nStep = 0;
		uchar* pMapData = NULL;
```

```
		cvGetRawData(pMap, &pMapData, &nStep, &szImage);
		nStep = nStep/sizeof(pMapData[0]);

		for( int nSegment = 0; nSegment < vecCracks[nCrack].m_vecCrackSegments.size();
nSegment++ )
			for(int nPoint = 0; nPoint <
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints.size(); nPoint++ )
			{
				CrackPoint pt =
vecCracks[nCrack].m_vecCrackSegments[nSegment].m_vecCrackPoints[nPoint];

				pMapData[pt.m_nY*nStep + pt.m_nX] = nSegment;
			}
	}


	void  CCrackDetection:: UpdateSegmentOBB(CrackSegment& segment, CvMemStorage*
ptStorage)
	{
		if( ptStorage == NULL ) return;

		CvSeq *points = cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq),
sizeof(CvPoint), ptStorage );

		for( int nCrackPoint = 0; nCrackPoint < segment.m_vecCrackPoints.size();
nCrackPoint++ )
		{
			CrackPoint crack_point = segment.m_vecCrackPoints[nCrackPoint];

			CvPoint pt;
			pt.x = crack_point.m_nX; pt.y = crack_point.m_nY;
			cvSeqPush(points, &pt);
		}

		CvBox2D box = cvMinAreaRect2(points, ptStorage);

		if(box.size.height >= box.size.width)
			segment.m_cvOBB2D = box;
		else
		{
			segment.m_cvOBB2D.center = box.center;
			segment.m_cvOBB2D.size.width = box.size.height;
			segment.m_cvOBB2D.size.height = box.size.width;
			segment.m_cvOBB2D.angle = box.angle + 90;
		}

	}

	void CCrackDetection::DetectCrackSkeleton()
	{
		if( m_pCrackMap == NULL ) return;
		SAFE_RELEASE_IMAGE(m_pCrackSkeleton);

	  IplImage* src = m_pCrackMap;
	  m_pCrackSkeleton = cvCloneImage(src);
```

```
		// the following code is from Momma's website

		IplImage* src_w = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, 1);
		IplImage* src_b = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, 1);
		IplImage* src_f = cvCreateImage(cvGetSize(src), IPL_DEPTH_32F, 1);

		cvScale(src, src_f, 1/255.0, 0);

		cvThreshold(src_f,src_f,0.5,1.0,CV_THRESH_BINARY);
		cvThreshold(src_f,src_w,0.5,1.0,CV_THRESH_BINARY);
		cvThreshold(src_f,src_b,0.5,1.0,CV_THRESH_BINARY_INV);

		double sum=1;
		while(sum>0)
		{
		 sum=0;
		 for (int i=0; i<8; i++)
		 {
			cvFilter2D(src_w, src_w, *(kpw+i));
			cvFilter2D(src_b, src_b, *(kpb+i));
			cvThreshold(src_w,src_w,2.99,1,CV_THRESH_BINARY); //2.5->2.99
			cvThreshold(src_b,src_b,2.99,1,CV_THRESH_BINARY); //2.5->2.99
			cvAnd(src_w, src_b, src_w);
			sum += cvSum(src_w).val[0];
			cvXor(src_f, src_w, src_f);
			cvCopyImage(src_f, src_w);
			cvThreshold(src_f,src_b,0.5,1,CV_THRESH_BINARY_INV);
		 }
		}

		cvConvertScaleAbs(src_f, m_pCrackSkeleton, 255, 0);
	}

	void CCrackDetection::DetectCrackSegments()
	{
		if(m_pCrackSkeleton == NULL) return;

		int x = 0, y = 0;

		int NeighborCandidateX[8] = {-1, -1, -1,  0,  1, 1, 1, 0};
		int NeighborCandidateY[8] = { 1,  0, -1, -1, -1, 0, 1, 1};

		CvSize szImage;
		int    nStep = 0;
		uchar* pSkeletonData = NULL;
		cvGetRawData(m_pCrackSkeleton, &pSkeletonData, &nStep, &szImage);

		IplImage* pCrackMark = cvCreateImage(cvSize(m_pCrackSkeleton->width,
m_pCrackSkeleton->height),
			IPL_DEPTH_8U, 1);
		cvZero(pCrackMark);
		uchar* pCrackMarkData = NULL;
		cvGetRawData(pCrackMark, &pCrackMarkData);

		m_vecCracks.clear();
```

```
		Crack newCrack;
		newCrack.m_vecCrackSegments.reserve(50);
		newCrack.m_vecCrackSegments.clear();

		CrackSegment newSegment;
		newSegment.m_vecCrackPoints.reserve(100);
		newSegment.m_vecCrackPoints.clear();

		int nCrackSegment = 0;

		vector<CrackPoint> vecCandidatePoints;
		vecCandidatePoints.reserve(10);
		vecCandidatePoints.clear();

		CrackPoint seedPoint, neighborPoint;

		for( y = 0; y < szImage.height; y++ )
		for( x = 0; x < szImage.width; x++ )
		{
			if( pSkeletonData[y*nStep + x] == 0 || pCrackMarkData[y*nStep + x] != 0 )
continue;

			pCrackMarkData[y*nStep + x] = 1;

			// New cracks
			seedPoint.m_nY = y;
			seedPoint.m_nX = x;

			newSegment.m_vecCrackPoints.clear();
			newSegment.m_vecCrackPoints.push_back(seedPoint);

			newCrack.m_vecCrackSegments.clear();
			newCrack.m_vecCrackSegments.push_back(newSegment);
			nCrackSegment = 0;

			while( nCrackSegment < newCrack.m_vecCrackSegments.size() )
			{
				vecCandidatePoints.clear();

				for( int nNeighbor = 0; nNeighbor < 8; nNeighbor++ )
				{
					neighborPoint.m_nX = seedPoint.m_nX +
NeighborCandidateX[nNeighbor];
					neighborPoint.m_nY = seedPoint.m_nY +
NeighborCandidateY[nNeighbor];


					if( neighborPoint.m_nX < 0 || neighborPoint.m_nX >=
szImage.width ||
					   neighborPoint.m_nY < 0 || neighborPoint.m_nY >=
szImage.height )
						continue;

					if( pCrackMarkData[neighborPoint.m_nY*nStep +
neighborPoint.m_nX] != 0 ||
```

```
                                        pSkeletonData[neighborPoint.m_nY*nStep +
neighborPoint.m_nX] == 0 )
                                        continue;

                                vecCandidatePoints.push_back(neighborPoint);

                        }

                        int nSize = vecCandidatePoints.size();

                        if( nSize == 1 )
                        {

        newCrack.m_vecCrackSegments[nCrackSegment].m_vecCrackPoints.push_back(vecCandidatePo
ints[0]);
                                seedPoint = vecCandidatePoints[0];

                                pCrackMarkData[vecCandidatePoints[0].m_nY*nStep +
vecCandidatePoints[0].m_nX] = 1;
                        }

                        else
                        {
                                for( int nSegment = 0; nSegment < nSize; nSegment++ )
                                {
                                        newSegment.m_vecCrackPoints.clear();

        newSegment.m_vecCrackPoints.push_back(vecCandidatePoints[nSegment]);

        newCrack.m_vecCrackSegments.push_back(newSegment);


        pCrackMarkData[vecCandidatePoints[nSegment].m_nY*nStep +
                                                        vecCandidatePoints[nSegment].m_nX] = 1;
                                }

                                nCrackSegment++;
                                if( nCrackSegment <  newCrack.m_vecCrackSegments.size())

                                   seedPoint =
newCrack.m_vecCrackSegments[nCrackSegment].m_vecCrackPoints[0];
                        }
                }

                m_vecCracks.push_back(newCrack);

        }

}

void CCrackDetection::DetectCrackDistance()
{
        if( m_pCrackMap == NULL ) return;

        m_pCrackDistance = cvCreateImage(cvSize(m_pCrackMap->width, m_pCrackMap-
>height),
```

```
                    IPL_DEPTH_32F, 1);
                cvZero(m_pCrackDistance);

                cvDistTransform(m_pCrackMap, m_pCrackDistance, CV_DIST_L2, 0);

                //  Clear the boundary
                CvSize szImage = cvSize(m_pCrackDistance->width, m_pCrackDistance->height);
                int nStep = 0;
                float* pData = NULL;

                cvGetRawData(m_pCrackDistance, (uchar**)(&pData), &nStep, &szImage);
                nStep /= sizeof(pData[0]);

                int i = 0;
                for( i = 0; i < szImage.width; i++ )
                        pData[0*nStep + i] = pData[(szImage.height-1)*nStep + i] = 0.0f;

                for( i = 0; i < szImage.height; i++ )
                        pData[i*nStep + 0] = pData[i*nStep + szImage.width - 1 ] = 0.0f;

        }


        void CCrackDetection::ReduceCrackSegments(int nMinThreshSegmentSize, vector<Crack>&
vecCracks,
                vector<Crack>& vecSimplifiedCracks)
        {
                int nCrack = 0;
                int nSegment = 0;
                int nCandidate = 0;

                int nLargeSegment = 0;
                int nLargeSegmentSize = 0;

                vecSimplifiedCracks = vecCracks;

                Crack newCrack;
                newCrack.m_vecCrackSegments.reserve(100);
                newCrack.m_vecCrackSegments.clear();

                for( nCrack = 0; nCrack < vecSimplifiedCracks.size(); nCrack++ )
                {
                        SortCrackSegmentbySize(vecSimplifiedCracks, nCrack);

                        for( nLargeSegment = 0; nLargeSegment <
vecSimplifiedCracks[nCrack].m_vecCrackSegments.size(); nLargeSegment++ )
                        {
                                nLargeSegmentSize =
vecSimplifiedCracks[nCrack].m_vecCrackSegments[nLargeSegment].m_vecCrackPoints.size();

                                if( nLargeSegmentSize < nMinThreshSegmentSize) break;
                        }

                        vecSimplifiedCracks[nCrack].m_vecCrackSegments.erase(
                                vecSimplifiedCracks[nCrack].m_vecCrackSegments.begin() +
nLargeSegment,
```

```
                                vecSimplifiedCracks[nCrack].m_vecCrackSegments.end());

                        MergeCrackPoints(nCrack, vecSimplifiedCracks);

                        SortCrackSegmentbySize(vecSimplifiedCracks, nCrack);
                }
        }


        void  CCrackDetection:: SortCrackSegmentbyDistance(vector<CrackSegment>& vecSegments,
CvPoint2D32f center)
        {
                int i, j, flag = 1;                             // set flag to 1 to begin initial pass
                int segmentLength = vecSegments.size();

                for(i = 0; (i < segmentLength) && flag; i++)
                {
                        flag = 0;
                        for (j = 0; j < (segmentLength -1 - i); j++)
                        {
                                CvPoint2D32f point_j = vecSegments[j].m_cvOBB2D.center;
                                CvPoint2D32f point_j_1 = vecSegments[j+1].m_cvOBB2D.center;

                                if ( EculideanDistance(center, point_j) > EculideanDistance(center,
point_j_1) )
                                {
                                        SwapCrackSegment(vecSegments, j, j+1);
                                        flag = 1;
        // indicates that a swap occurred.
                                }
                        }
                }

                return;
        }

        void  CCrackDetection:: SortCrackSegmentbyOrientation(vector<Crack>& vecCracks, int nCrack)
        {
                int i, j, flag = 1;                             // set flag to 1 to begin initial pass
                int segmentLength = vecCracks[nCrack].m_vecCrackSegments.size();

                for(i = 0; (i < segmentLength) && flag; i++)
                {
                        flag = 0;
                        for (j = 0; j < (segmentLength -1 - i); j++)
                        {
                                if
(vecCracks[nCrack].m_vecCrackSegments[j+1].m_cvOBB2D.angle >

        vecCracks[nCrack].m_vecCrackSegments[j].m_cvOBB2D.angle)
                                {
                                        SwapCrackSegment(vecCracks, nCrack, j, j+1);          // swap
elements
                                        flag = 1;
        // indicates that a swap occurred.
                                }
```

```
                }
        }

        return;
}

void  CCrackDetection:: SortCrackSegmentbySize(vector<Crack>& vecCracks, int nCrack)
{
        int i, j, flag = 1;                             // set flag to 1 to begin initial pass
        int segmentLength = vecCracks[nCrack].m_vecCrackSegments.size();

        for(i = 0; (i < segmentLength) && flag; i++)
        {
                flag = 0;
                for (j = 0; j < (segmentLength -1 - i); j++)
                {
                        if
(vecCracks[nCrack].m_vecCrackSegments[j+1].m_vecCrackPoints.size() >

vecCracks[nCrack].m_vecCrackSegments[j].m_vecCrackPoints.size())
                        {
                                SwapCrackSegment(vecCracks, nCrack, j, j+1);          // swap
elements
                                flag = 1;
// indicates that a swap occurred.
                        }
                }
        }

        return;   //arrays are passed to functions by address; nothing is returned

}


void  CCrackDetection:: SwapCrackSegment(vector<CrackSegment>& vecSegments, int
nSegment1, int nSegment2)
{
        CrackSegment segmentTemp = vecSegments[nSegment1];
        vecSegments[nSegment1] = vecSegments[nSegment2];
        vecSegments[nSegment2] = segmentTemp;

}

void  CCrackDetection:: SwapCrackSegment(vector<Crack>& vecCracks, int nCrack, int
nSegment1, int nSegment2)
{
        CrackSegment segmentTemp = vecCracks[nCrack].m_vecCrackSegments[nSegment1];
        vecCracks[nCrack].m_vecCrackSegments[nSegment1] =
vecCracks[nCrack].m_vecCrackSegments[nSegment2];
        vecCracks[nCrack].m_vecCrackSegments[nSegment2] = segmentTemp;

}

void CCrackDetection:: PrintCrackProperties(const char* filename)
{
        FILE* fp = NULL;
```

```
		fp = fopen(filename, "w");

		if(fp == NULL) return;

		int nCrack;
		int nSegment;

		fprintf(fp,
"*********************************************************************************\n");
		fprintf(fp, "\n");
		fprintf(fp, "# of cracks: %d\n", m_vecCrackProperties.size());

		for( nCrack = 0; nCrack < m_vecCrackProperties.size(); nCrack++ )
		{
			fprintf(fp, "    Crack No. %d\n", nCrack+1);

			int nSegmentSize = m_vecCrackProperties[nCrack].m_vecSegProperties.size();

			fprintf(fp, "    # of segments: %d\n", nSegmentSize);

			for ( nSegment = 0; nSegment < nSegmentSize; nSegment++ )
			{
				SegmentProperty segProperty =
m_vecCrackProperties[nCrack].m_vecSegProperties[nSegment];

				fprintf(fp, "\n");
				fprintf(fp, "       Segment No. %d:\n",  nSegment+1);
				fprintf(fp, "       Location: (%d, %d)  (%d, %d)\n",
					segProperty.m_ptLocations[0].x,
segProperty.m_ptLocations[0].y,
					segProperty.m_ptLocations[1].x,
segProperty.m_ptLocations[1].y);
				fprintf(fp, "       Length: %8.3f\n", segProperty.m_fLength);
				fprintf(fp, "       Orientation: %8.3f\n", segProperty.m_fOrientation);
				fprintf(fp, "       Avg width: %8.3f\n", segProperty.m_fAvgWidth);
				fprintf(fp, "       Max width: %8.3f\n", segProperty.m_fMaxWidth);

				fprintf(fp, "       Length2width: %8.3f\n",
segProperty.m_fLength2ColumnWidth);
				fprintf(fp, "       Avg width2width: %8.3f\n",
segProperty.m_fAvgWidth2ColumnWidth);
				fprintf(fp, "       Max width2width: %8.3f\n",
segProperty.m_fMaxWidth2ColumnWidth);
				fprintf(fp, "\n");
			}

			fprintf(fp, "\n");
		}

		fprintf(fp,
"*********************************************************************************\n");

		SAFE_RELEASE_FILE(fp);
	}

}
```

APPENDIX C: CODE FOR AIR POCKETS/DISOCLORATION DETECTION AND PROPERTIES RETRIEVAL

This appendix presents the main part in the prototype code for the detection and properties retrieval of air pockets and discoloration.

```
/*********************************************************
// Macro definition for max, min, lessequal, and distance
*********************************************************/

#define ZMAX(x,y) ((x)>=(y)?(x):(y))
#define ZMIN(x,y) ((x)<=(y)?(x):(y))
#define ZLESSEQUAL(r0,r) ((r0)<=(r)?TRUE:FALSE)
#define ZDISTANCE(x,y,x0,y0) (sqrt((float)(((x)-(x0))*((x)-(x0))+((y)-(y0))*((y)-(y0)))))

BOOL isLocalMaximum( int x, int y, int size, IplImage* pImage, float* pIntensity)
{
        int i, j;
        int step;
        float fData;

        float* pImageData = NULL;
        cvGetRawData( pImage, (uchar**)&pImageData,&step);
        step /= sizeof(pImageData[0]);

        fData = (float) pImageData[y*step+x];

        if( fData < intensityThreshold ) return FALSE;

        float sizex,sizey, mean = 0;

        for( j = ZMAX(y - size,0); j < ZMIN(y + size,pImage->height); j++)
        for( i = ZMAX(x - size,0); i < ZMIN(x + size,pImage->width); i++)
        {
                mean += pImageData[j*step + i];
                if(j == y && i == x) continue;
                if( pImageData[j*step + i] > fData )
                        return FALSE;
        }

        sizey = ZMIN(y + size,pImage->height) - ZMAX(y - size,0);
        sizex = ZMIN(x + size,pImage->width) - ZMAX(x - size,0);
        mean /= (sizey*sizex);

        (*pIntensity) = pImageData[y*step+x];
        return TRUE;
}

void getImageIntensity(IplImage* pSrc, IplImage* pDst, BOOL bInv)
{
```

```
	if( pSrc == NULL || pDst == NULL ) return;

	if( (pSrc->nChannels == 3 && pDst->nChannels == 1) &&
		(pSrc->depth = IPL_DEPTH_8U && pDst->depth == IPL_DEPTH_32F) )
	{

		uchar* pImageSrc = NULL;
		float* pImageDst = NULL;

		int srcStep, dstStep;
		CvSize size;
		int x, y;

		cvGetRawData( pSrc, (uchar**)&pImageSrc, &srcStep, &size );
		srcStep /= sizeof(pImageSrc[0]);

		cvGetRawData( pDst, (uchar**)&pImageDst, &dstStep);
		dstStep /= sizeof(pImageDst[0]);

		for( y = 0; y < size.height; y++, pImageSrc += srcStep )
		for( x = 0; x < size.width; x++ )
		{
			pImageDst[y*dstStep + x] = (float) ((pImageSrc[pSrc->nChannels*x] +
				pImageSrc[pSrc->nChannels*x+1] +
				pImageSrc[pSrc->nChannels*x+2]) / 3.0 );

			if(bInv == FALSE)
				pImageDst[y*dstStep + x] /= 255.0f;
			else
				pImageDst[y*dstStep + x] = 1.0f - pImageDst[y*dstStep + x]/255.0f; //
Flip
		}
	}
}


void getRegionAvgSdv(IplImage* pImage, int nX, int nY, int nWidth, int nHeight,
                                        float* pAvg, float* pSdv)
{
	IplImage* pRegion = NULL;
	float avg = 0, sdv = 0;;
	int step;
	CvSize size;
	float* pImageData = NULL;

	cvGetRawData( pImage, (uchar**)&pImageData, &step, &size);
	step /= sizeof(pImageData[0]);

	int x, y;

	for( y = nY; y < nY+nHeight; y++)
  for( x = nX; x < nX+nWidth; x++ )
		avg += pImageData[y*step + x];

	avg /= (nWidth*nHeight);
```

```
        for( y = nY; y < nY+nHeight; y++)
    for( x = nX; x < nX+nWidth; x++ )
                sdv += (pImageData[y*step + x] - avg)*(pImageData[y*step + x] - avg);

        sdv = sqrt(sdv/(nWidth*nHeight));

        (*pAvg) = avg;
        (*pSdv) = sdv;
}

void getSelectedImage(IplImage** ppSelected, IplImage* pSrc, INT nX, INT nY, INT nWidth, INT
nHeight)
{
        (*ppSelected) = NULL;

        INT nImageWidth, nImageHeight;

        nImageWidth = pSrc->width;
        nImageHeight = pSrc->height;

        cvSetImageROI(pSrc, cvRect(nX, nY, nWidth, nHeight));
        (*ppSelected) = cvCreateImage(cvGetSize(pSrc), pSrc->depth, pSrc->nChannels);
        (*ppSelected)->origin = pSrc->origin;
        cvCopy(pSrc, (*ppSelected), NULL);

        cvResetImageROI(pSrc);
}

void adjustImage(IplImage* pSrc, float min, float max)
{
        float fMin = 0, fMax = 0;;
        int step;
        CvSize size;
        float* pImageData = NULL;

        cvGetRawData( pSrc, (uchar**)&pImageData, &step, &size);
        step /= sizeof(pImageData[0]);

        int x, y;
        fMin = fMax = (float)(pImageData[0]);
        for( y = 0; y < size.height; y++)
    for( x = 0; x < size.width; x++ )
        {
                if( pImageData[y*step + x] < fMin ) fMin = pImageData[y*step + x];
                else if ( pImageData[y*step + x] > fMax ) fMax = pImageData[y*step + x];
        }

        float fInv = 1.0/(fMax - fMin);

        for( y = 0; y < size.height; y++)
    for( x = 0; x < size.width; x++ )
                pImageData[y*step + x] = min + (pImageData[y*step + x] - fMin)*fInv*(max-min);

}
```

```
void CRegion::CalculateSignature()
{
	float rm = 0, bm = 0, gm = 0, im = 0;

	PixelPoint ppoint;
	int size = m_vecPoints.size();

	for(int i = 0; i < size; i++)
	{
		ppoint = m_vecPoints[i];

		rm += ppoint.m_fRed;
		bm += ppoint.m_fBlue;
		gm += ppoint.m_fGreen;
	}

	rm /= size;
	gm /= size;
	bm /= size;

	im = (rm + gm + bm);

	rm = rm/im * 100;   // normalized red percentage – color characteristics
	gm = gm/im * 100;   // normalized green percentage – color characteristics
	bm = bm/im * 100;   // normalized blue percentage – color characteristics

	im = im/(255*3)*100;

	m_fIntensityMean = im;
	m_fRedMean = rm;
	m_fGreenMean = gm;
	m_fBlueMean = bm;
}


void CRegion::InsertPoints(INT nX, INT nY, float red, float green, float blue)
{
	PixelPoint ppoint;
	ppoint.m_nX = nX;
	ppoint.m_nY = nY;
	ppoint.m_fRed = red;
	ppoint.m_fGreen= green;
	ppoint.m_fBlue = blue;

	m_vecPoints.push_back(ppoint);
}

void CConcreteInspectionDoc::CreatePyramid(IplImage* pSrc)
{
	if( pSrc == NULL || pSrc->depth != IPL_DEPTH_32F)		return;

	IplImage* pImageIntensity = pSrc;

	ClearPyramid();

	(m_pPyrImage[0]) = cvCreateImage(cvSize((pImageIntensity->width),(pImageIntensity->height)),
```

```
                pImageIntensity->depth,pImageIntensity->nChannels);
        cvCopy(pImageIntensity, m_pPyrImage[0]);            // Original one

        float fPercentage;
        CvSize size = cvSize(m_pPyrImage[0]->width, m_pPyrImage[0]->height);

        for( int i = 1; i < m_nLevels; i++)
        {
                fPercentage = m_fPercentages[i];
                (m_pPyrImage[i]) = cvCreateImage(
                        cvSize((int)((size.width)*fPercentage),(int)((size.height)*fPercentage)),
                        pImageIntensity->depth,pImageIntensity->nChannels);

                cvResize(m_pPyrImage[0], m_pPyrImage[i], CV_INTER_AREA);      // Pyramid down;

        }

        pImageIntensity = NULL;
}

void CConcreteInspectionDoc::CalculateGradients(IplImage* pXDer, IplImage* pYDer, IplImage**
ppGradient)
{
        (*ppGradient) = NULL;
        if(pXDer->width != pYDer->width || pXDer->height != pYDer->height) return;

        (*ppGradient) = cvCreateImage(cvSize(pXDer->width, pXDer->height),
                pXDer->depth, pXDer->nChannels);
        cvZero((*ppGradient));

        float* pXData = NULL;
        float* pYData = NULL;
        float* pGradientData = NULL;

        int step;
        CvSize size;

        cvGetRawData(pXDer, (uchar**)(&pXData));
        cvGetRawData(pYDer, (uchar**)(&pYData));
        cvGetRawData((*ppGradient), (uchar**)(&pGradientData), &step, &size);

        step /= sizeof(pGradientData[0]);

        for(int y = 0; y < size.height; y++)
        for(int x = 0; x < size.width; x++)
        {
                float dx = pXData[y*step+x];
                float dy = pYData[y*step+x];

                pGradientData[y*step+x] = sqrt(dx*dx+dy*dy);
        }
}

void CConcreteInspectionDoc::CalculateDerivatives(IplImage* pSrc, IplImage** ppXDer, IplImage**
ppYDer)
{
```

```
        (*ppXDer) = cvCreateImage(cvSize(pSrc->width,pSrc->height), pSrc->depth, pSrc->nChannels);
        (*ppYDer) = cvCreateImage(cvSize(pSrc->width,pSrc->height), pSrc->depth, pSrc->nChannels);
        cvSobel(pSrc,(*ppXDer),1,0);
        cvSobel(pSrc,(*ppYDer),0,1);
}

inline float f(float t) { return t > 0.008856? pow(t, 1.0f/3): 7.787*t+16.0f/116; }


void CConcreteInspectionDoc::DetectColorUniform(INT nX, INT nY, INT nWidth, INT nHeight, float&
fVIR1, float& fVIR2)
{
        // Show discoloration
        IplImage* pSegmentation = cvCloneImage(pSelectedImage);
        // Graph cut segmentation
        int num_ccs;
        CGraphCutSeg graphCut;

        graphCut.Initialize(pSelectedImage);
        graphCut.SegmentImage(0.8f, 100, 80, &num_ccs);

        graphCut.GetSegmentationResult(nX, nY, &m_vecRegion);
        graphCut.GetSegmentationImage(&pSegmentation);

//  Identify discoloration regions through region growing
   CRegion region;
        int regionnum;
        int row;
        float red, green, blue, intensity;
        float sred, sgreen, sblue, sintensity;
        float similarity;

        int sregion = -1;
        int sarea = -1;

        float mintensity = 0.0f;

        for(regionnum = 0; regionnum < m_vecRegion.size(); regionnum++)
        {
                region = m_vecRegion[regionnum];
                mintensity += region.GetIntensity();
                if(region.GetPixelPointsize() > sarea)
                        {
                        sarea = region.GetPixelPointsize();
                        sregion = regionnum;
                        }
        }

        mintensity /= m_vecRegion.size();
   m_vecRegion[sregion].GetSignature(sintensity, sred, sgreen, sblue);

        float percentage = 0;
   float totalsize = 0;

        for(regionnum = 0; regionnum < m_vecRegion.size(); regionnum++)
        {
```

```
	region = m_vecRegion[regionnum];

			region.GetSignature(intensity, red, green, blue);



				similarity = ColorSimilarity(red, green, blue, sred, sgreen, sblue);

			if(similarity <= similarityThreshold)
				{
					m_vecRegion[regionnum].SetMaterial(1);  // Concrete
				}
				else
				{
			    percentage += m_vecRegion[regionnum].GetPixelPointsize();
				  m_vecRegion[regionnum].SetMaterial(2);  // Discoloration
				}

			totalsize += m_vecRegion[regionnum].GetPixelPointsize();
		}

		if(percentage > 0.5*totalsize)
			{
				for(regionnum = 0; regionnum < m_vecRegion.size(); regionnum++)
					{
					region = m_vecRegion[regionnum];
					if(region.GetMaterial() == 1)
						m_vecRegion[regionnum].SetMaterial(2);
					else
						m_vecRegion[regionnum].SetMaterial(1);
					}
			}

		CalculateVIRD(fVIR1, fVIR2);

		UpdateCvvImage(2); // 2 -- discoloration


		cvReleaseImage(&pSegmentation); pSegmentation = NULL;
		cvReleaseImage(&pIntensity); pIntensity = NULL;
		cvReleaseImage(&pSelectedImage); pSelectedImage = NULL;

}

void CConcreteInspectionDoc::Segmentation(INT nX, INT nY, INT nWidth, INT nHeight)
{
		IplImage *pSelectedImage = NULL;
		IplImage *pSegmentation = NULL;

		getSelectedImage(&pSelectedImage, m_pIplOrigImage, nX, nY, nWidth, nHeight);
		if(pSelectedImage == NULL) return;

		pSegmentation = cvCloneImage(pSelectedImage);

		// Graph cut segmentation
		int num_ccs;
```

```
	CGraphCutSeg graphCut;

	graphCut.Initialize(pSelectedImage);
	graphCut.SegmentImage(0.8f, 100, 80, &num_ccs);
	graphCut.GetSegmentationImage(&pSegmentation);

	showImage(pSegmentation);

	cvReleaseImage(&pSegmentation); pSegmentation = NULL;
	cvReleaseImage(&pSelectedImage); pSelectedImage = NULL;
}

void CConcreteInspectionDoc::Gradient(INT nX, INT nY, INT nWidth, INT nHeight)
{
	IplImage *pSelectedImage = NULL;

	getSelectedImage(&pSelectedImage, m_pIplOrigImage, nX, nY, nWidth, nHeight);
	if(pSelectedImage == NULL) return;


	IplImage* pXDerivative = NULL;
	IplImage* pYDerivative = NULL;
	IplImage* pGradient = NULL;

	IplImage* pIntensity = cvCreateImage(cvSize((pSelectedImage->width),(pSelectedImage-
>height)),
				IPL_DEPTH_32F, 1);  // Intensity of image. float;

	getImageIntensity(pSelectedImage, pIntensity, TRUE);
	CalculateDerivatives(pIntensity, &pXDerivative, &pYDerivative);
	CalculateGradients(pXDerivative,pYDerivative,&pGradient);

	showImage(pGradient);

	cvReleaseImage(&pIntensity); pIntensity = NULL;
	cvReleaseImage(&pXDerivative); pXDerivative = NULL;
	cvReleaseImage(&pYDerivative); pYDerivative = NULL;
	cvReleaseImage(&pGradient); pGradient = NULL;
	cvReleaseImage(&pSelectedImage); pSelectedImage = NULL;
}

void CConcreteInspectionDoc::DetectAirPocket(INT nX, INT nY, INT nWidth, INT nHeight, float&
fVIR1, float& fVIR2)
{
	IplImage *pSelectedImage = NULL;
	IplImage *pIntensity = NULL;
	IplImage* pSmooth = NULL;

	getSelectedImage(&pSelectedImage, m_pIplOrigImage, nX, nY, nWidth, nHeight);
	if(pSelectedImage == NULL) return;

	pSmooth = cvCloneImage(pSelectedImage);
	cvSmooth( pSelectedImage, pSmooth, CV_MEDIAN, 3, 0, 1, 0 );

	pIntensity = cvCreateImage(cvSize(pSelectedImage->width, pSelectedImage->height),
		IPL_DEPTH_32F, 1);
```

```
		getImageIntensity(pSmooth, pIntensity, TRUE);

		adjustImage(pIntensity, 0, 1);

		CreatePyramid(pIntensity);

		CvMat* pFilter = NULL;
		LoadFilter("E:\\Zhenhua\\Codes\\ConcreteInspection\\filter.xml", "Spot5x5", &pFilter);

		m_vecAPMarked.clear();

		int nRadius = 0;
		float fPercentage;
		for(int nLevel = m_nLevels - 1; nLevel >=0; nLevel--)
		{
			fPercentage = m_fPercentages[nLevel];

			IplImage* pResult = NULL;

			DetectAirPocket(nLevel, pFilter, &pResult);

			nRadius = (int)((pFilter->cols)/(2*fPercentage));

			FindAirPocket(nX, nY, nLevel, nRadius, pResult);

			cvReleaseImage(&pResult);
			pResult = NULL;
		}

		char filename[256];

		sprintf(filename,"%s.txt", m_strFileName);
		PrintOutPockets(filename, pSelectedImage->width*pSelectedImage->height);

		CalculateVIRAP(fVIR1, fVIR2, pSelectedImage->width*pSelectedImage->height);

		cvReleaseMat(&pFilter);
		pFilter = NULL;


		cvReleaseImage(&pIntensity); pIntensity = NULL;
		cvReleaseImage(&pSmooth); pSmooth = NULL;

		UpdateCvvImage(1); // 1 -- Air pockets
}

void CConcreteInspectionDoc::Histogram(INT nX, INT nY, INT nWidth, INT nHeight)
{
		IplImage* pSelectedImage = NULL;
		IplImage* pImageIntensity = NULL;

		getSelectedImage(&pSelectedImage, m_pIplOrigImage, nX, nY, nWidth, nHeight);
		if(pSelectedImage == NULL) return;
		pImageIntensity = cvCreateImage( cvGetSize(pSelectedImage), 8, 1 );
		cvCvtColor(pSelectedImage, pImageIntensity, CV_BGR2GRAY);
```

```
        int gray_bins = 256;
    int hist_size[] = {gray_bins};
    float gray_range[] = { 0, 1 };
    float *ranges[] = {gray_range};
    CvHistogram* hist;
    float max_value = 0;

    hist = cvCreateHist(1, hist_size, CV_HIST_ARRAY); // , ranges, 1 );
    cvCalcHist(&pImageIntensity, hist, 0, 0 );
    cvGetMinMaxHistValue( hist, 0, &max_value, 0, 0 );
    IplImage* hist_img = cvCreateImage( cvSize(gray_bins,256), 8, 3 );
    cvSet( hist_img, CV_RGB(255,255,255) );

    for(int g = 0; g < gray_bins; g++ )
    {
                float bin_val = cvQueryHistValue_1D( hist, g);
       int intensity = cvRound(bin_val*255/max_value);
       cvRectangle( hist_img, cvPoint( g, 256*0.8),
            cvPoint((g+1) - 1, (256 - intensity)*0.8),
            CV_RGB(0,0,0),
            CV_FILLED );
    }

        showImage(hist_img);
        cvReleaseImage(&hist_img);
        cvReleaseImage(&pImageIntensity);
        cvReleaseImage(&pSelectedImage);
        cvReleaseHist(&hist);
}

void CConcreteInspectionDoc::LoadFilter(const char* strFilename, const char* strFilter, CvMat**
ppResponse)
{
        (*ppResponse) = NULL;
        if ( strFilter == NULL ) return;

        CvMat* pMatFilter = NULL;

        CvFileStorage* myFileStorage = cvOpenFileStorage(strFilename, NULL,
CV_STORAGE_READ);

        CvFileNode* myNodeRoot = cvGetRootFileNode( myFileStorage);
        CvFileNode* myNode = cvGetFileNodeByName( myFileStorage, myNodeRoot, strFilter );
        pMatFilter = (CvMat*) cvRead( myFileStorage, myNode);

        cvReleaseFileStorage( &myFileStorage );
        (*ppResponse) = pMatFilter;
}

void CConcreteInspectionDoc:: DetectAirPocket(int nLevel, CvMat* pFilter, IplImage** ppResult)
{
        (*ppResult) = NULL;

        if( m_pPyrImage[nLevel] == NULL || pFilter == NULL ) return;
```

```
	(*ppResult) = cvCreateImage(cvSize(m_pPyrImage[nLevel]->width, m_pPyrImage[nLevel]-
>height),
			m_pPyrImage[nLevel]->depth, m_pPyrImage[nLevel]->nChannels);

	cvFilter2D(m_pPyrImage[nLevel], (*ppResult), pFilter, cvPoint(pFilter->rows/2+1, pFilter-
>cols/2+1));

	float* pImageData = NULL;
	int step;
	CvSize size;

	cvGetRawData( (*ppResult), (uchar**)&pImageData, &step, &size);
	step /= sizeof(pImageData[0]);

	float maxValue = pImageData[0];

	for( int y = 0; y < size.height; y++ )
	for( int x = 0; x < size.width; x++ )
	{
		if( pImageData[y*size.width+x] < 0 ) pImageData[y*size.width+x] = 0;
		if( maxValue < pImageData[y*size.width+x] ) maxValue = pImageData[y*size.width+x];
	}

	if( maxValue != 0 )
	{
		float maxValueInv = 1.0f/maxValue;

		for( int y = 0; y < size.height; y++ )
		for( int x = 0; x < size.width; x++ )
			pImageData[y*size.width+x] *= maxValueInv;
	}

}


void CConcreteInspectionDoc::FindAirPocket(int nShiftX, int nShiftY, int nLevel, int nRadius, IplImage*
pImage)
{
	if(pImage == NULL) return;

	int x, y;
	int nWidth, nHeight;
	float fIntensity;
	float fPercentage = m_fPercentages[nLevel];

	nWidth = pImage->width;
	nHeight = pImage->height;

	for( y = 2; y < nHeight-2; y++ )
	for( x = 2; x < nWidth-2; x++ )
	{
		if ( isLocalMaximum(x, y, 5, pImage, &fIntensity) == TRUE )
		{
			RecordAirPockets(nShiftX + (int)(x/fPercentage),
				nShiftY + (int)(y/(fPercentage)),
				nLevel, nRadius, fIntensity);
```

```
                }
        }
}

void CConcreteInspectionDoc:: RecordAirPockets(int x, int y, int nLevel, int nRadius, float fIntensity)
{
        int i;
        float r0;
        AirPocket ap;

        for(i = 0; i < m_vecAPMarked.size(); i++)
        {
                ap = m_vecAPMarked[i];

                r0 = ZDISTANCE(x, y, ap.m_nPosX, ap.m_nPosY );

                if( r0 <= nRadius + ap.m_nRadius )
                {
                        if( ap.m_fIntensity < fIntensity )
                        {
                                m_vecAPMarked[i].m_nPosX = x;
                                m_vecAPMarked[i].m_nPosY = y;
                                m_vecAPMarked[i].m_nRadius = nRadius;
                                m_vecAPMarked[i].m_nLevel = nLevel;
                                m_vecAPMarked[i].m_fIntensity = fIntensity;
                        }
                        break;
                }
        }

        if( i == m_vecAPMarked.size() )
        {
                AirPocket ap;
                ap.m_nPosX = x;
                ap.m_nPosY = y;
                ap.m_nRadius = nRadius;
                ap.m_nLevel = nLevel;
                ap.m_fIntensity = fIntensity;
                m_vecAPMarked.push_back(ap);
        }
}


void CConcreteInspectionDoc::PrintOutPockets(const char* strFile, float totalArea)
{
        int i;
        AirPocket ap;
        float area = 0;
        int numbers[3];
        FILE* fp = fopen(strFile,"w");

        numbers[0] = numbers[1] = numbers[2] = 0;
        for( i = 0; i < m_vecAPMarked.size(); i++ )
        {
                ap = m_vecAPMarked[i];
```

```
                    fprintf(fp,"%d    %d    %d    %d\n",ap.m_nPosX, ap.m_nPosY, ap.m_nLevel,
2*ap.m_nRadius);
                    if(ap.m_nLevel == 0) numbers[0]++;
                    else if(ap.m_nLevel == 1) numbers[1]++;
                    else if(ap.m_nLevel == 2) numbers[2]++;

                    area += 3.14f*ap.m_nRadius*ap.m_nRadius;
          }
          fprintf(fp,"Level0: number=%d\n", numbers[0]);
          fprintf(fp,"Level1: number=%d\n", numbers[1]);
          fprintf(fp,"Level2: number=%d\n", numbers[2]);
          fprintf(fp,"Total: number=%d area=%f\n",m_vecAPMarked.size(), area);
          fprintf(fp,"Total: ratio1=%8.2f, ratio2=%8.2f\n", area/totalArea,
1000*area/(totalArea*m_vecAPMarked.size()));
          fclose(fp); fp = NULL;
}

void CConcreteInspectionDoc::CalculateVIRAP(float& fVIR1, float& fVIR2, float totalArea)
{
   int i;
          AirPocket ap;
          float area = 0;
          int numbers[3];

          numbers[0] = numbers[1] = numbers[2] = 0;
          for( i = 0; i < m_vecAPMarked.size(); i++ )
          {
                    ap = m_vecAPMarked[i];
                    area += 3.14f*ap.m_nRadius*ap.m_nRadius;
          }

          fVIR1 = area/totalArea;
   fVIR2 = area/(totalArea*m_vecAPMarked.size());

}

void CConcreteInspectionDoc::CalculateVIRD(float& fVIR1, float& fVIR2)
{

   CRegion region;
          int regionnum = 0;

          float fIntensity = 0, fRed = 0, fGreen = 0, fBlue = 0;

          float fRedDiscoloration = 0, fGreenDiscoloration = 0, fBlueDiscoloration = 0;
   float fRedNormal = 0, fGreenNormal = 0, fBlueNormal = 0;

          int iNumberDiscoloration = 0;
   int iNumberNormal = 0;

          float fSizeDiscoloration = 0;
          float fSizeNormal = 0;

          for(regionnum = 0; regionnum < m_vecRegion.size(); regionnum++)
          {
                    region = m_vecRegion[regionnum];
```

```
                region.GetSignature(fIntensity, fRed, fGreen, fBlue);

                if(region.GetMaterial() == 2)  // Discoloration
                {

                        fSizeDiscoloration += region.GetPixelPointsize();
                        fRedDiscoloration += (fRed*fSizeDiscoloration);
                        fGreenDiscoloration += (fGreen*fSizeDiscoloration);
                        fBlueDiscoloration += (fBlue*fSizeDiscoloration);

                        iNumberDiscoloration += fSizeDiscoloration;
                }
                else                    // Normal
                {
                        fSizeNormal += region.GetPixelPointsize();
                        fRedNormal += (fRed*fSizeNormal);
                        fGreenNormal += (fGreen*fSizeNormal);
                        fBlueNormal += (fBlue*fSizeNormal);

                        iNumberNormal += fSizeNormal;
                }
        }

        // Average color characteristis for discoloration regions
        fRedDiscoloration /= iNumberDiscoloration;
        fBlueDiscoloration /= iNumberDiscoloration;
        fGreenDiscoloration /= iNumberDiscoloration;

        // Average color characteristis for normal regions
        fRedNormal /= iNumberNormal;
        fBlueNormal /= iNumberNormal;
        fGreenNormal /= iNumberNormal;


        fVIR1 = (float)sqrt((float)((fRedDiscoloration-fRedNormal)*(fRedDiscoloration-fRedNormal) +
                (fGreenDiscoloration-fGreenNormal)*(fGreenDiscoloration-fGreenNormal) +
                (fBlueDiscoloration-fBlueNormal)*(fBlueDiscoloration-fBlueNormal) ));

        fVIR2 = (float)( iNumberDiscoloration)/ (iNumberNormal+ iNumberDiscoloration);
}


void CConcreteInspectionDoc::MarkAirPocket(IplImage* pImage)
{
        if ( pImage == NULL ) return;

        int i = 0;
        int thickness = 1;
        AirPocket ap;

        for( i = 0; i < m_vecAPMarked.size(); i++ )
        {
                ap = m_vecAPMarked[i];
                cvCircle(pImage, cvPoint(ap.m_nPosX,ap.m_nPosY), ap.m_nRadius,
CV_RGB(255,255,255), -1);
```

```
                cvCircle(pImage, cvPoint(ap.m_nPosX,ap.m_nPosY), ap.m_nRadius, CV_RGB(255,0,0),
thickness);
        }
}

void CConcreteInspectionDoc::MarkDiscoloration(IplImage* pImage)
{
        if( pImage == NULL ) return;

        CRegion region;
        PixelPoint ppoint;

        int i = 0, j = 0;
        int thickness = 1;
        int radius = 2;

        float alpha = 0.3f;
        float red, green, blue;
        float showred, showgreen, showblue;

        CvSize size;
        int step;
        uchar* values = NULL;
        cvGetRawData(pImage, (uchar**)&values, &step, &size);
        step /= sizeof(values[0]);

        showred = 255; showgreen = 0; showblue = 0;

        for( i = 0; i < m_vecRegion.size(); i++ )
        {
                region = m_vecRegion[i];
                if(region.GetMaterial() == 2)  // Discoloration
                {
                        for( j = 0; j < region.GetPixelPointsize(); j++ )
                        {
                                ppoint = region.GetPixelPoint(j);

                                red = values[ppoint.m_nY*step + 3*ppoint.m_nX + 0];
                                green = values[ppoint.m_nY*step + 3*ppoint.m_nX + 1];
                                blue = values[ppoint.m_nY*step + 3*ppoint.m_nX + 2];

                                values[ppoint.m_nY*step + 3*ppoint.m_nX + 2] = (uchar)(red*(1-
alpha) + alpha*showred);
                                values[ppoint.m_nY*step + 3*ppoint.m_nX + 1] = (uchar)(green*(1-
alpha) + alpha*showgreen);
                                values[ppoint.m_nY*step + 3*ppoint.m_nX + 0] = (uchar)(blue*(1-
alpha) + alpha*showblue);

                        }
                }
        }
}

void CConcreteInspectionDoc::UpdateCvvImage(INT nFlag)
{
        IplImage* pImage = cvCloneImage(m_pIplOrigImage);
```

```
        if(nFlag == 1)  // Air pockets
                MarkAirPocket(pImage);
        else if(nFlag == 2)             // Discoloration
                MarkDiscoloration(pImage);

        m_pCvvImage = new CvvImage;
        m_pCvvImage->CopyOf(pImage);
//      showImage(pImage);
        cvReleaseImage(&pImage);
        pImage = NULL;
}
```

REFERENCES

American Association of State and Highway Transportation Officials (AASHTO), (2001). “Manual for condition evaluation of bridges.” 2nd Ed. Washington, D.C.

Abolghasemi, V. and Ahmadyfard, A., (2008) "An Edge-Based Color-Aided Method for License Plate Detection" Image and Vision Computing Journal, Elsevier, In Press, Corrected Proof, 2008

Abdel-Qader, I., Pashaie-Rad, S., Abudayyeh, O., and Yehia, S. (2006). “PCA-Based algorithm for unsupervised bridge crack detection.” Advances in Engineering Software, 2006, 37(12): 771-778

Abdel-Qader, I., Abudayyeh, O., and Kelly, M. E. (2003). “Analysis of Edge-Detection Techniques for Crack Identification in Bridges.” Journal of Computation in Civil Engineering., Volume 17, Issue 4, pp. 255-263

Abudayyeh, O.Y, (1997). "Audio/Visual Information in Construction Project Control," Journal of Advances in Engineering Software, Volume 28, Number 2, March, 1997

ACI 116R-00, (2005) “Cement and Concrete Terminology”, ACI Manual of Concrete Practice 2005

ACI 201.1R-92, (2005) “Guide for Make a Condition Survey of Concrete”, ACI Manual of Concrete Practice 2005

ACI 228.2R-98, (2005), “Nondestructive Test Methods for Evaluation of Concrete in Structures”, ACI Manual of Concrete Practice 2005

ACI 349.3R, (2005), “Evaluation of Existing Nuclear Safety-Related Concrete Structures”, ACI Manual of Concrete Practice 2005

Aldunate, R., Ochoa, S.F., Pena-Mora F. and Nussbaum, M. (2006), “Robust Mobile Ad Hoc Space for Collaboration to Support Disaster Relief Efforts Involving Critical Physical Infrastructure”, Journal of Computing in Civil Engineering, Vol. 20, No. 1, pp. 13-27

Almansa, A., Desolneux, A., and Vamech, S., (2003) "Vanishing Point Detection without Any A Priori Information", IEEE Transactions on Pattern Analysis and Machine Intelligence , Vol. 25, No. 4, pp.502-507.

American Association of State and Highway Transportation Officials (AASHTO), (2001). "Manual for condition evaluation of bridges." 2nd Ed. Washington, D.C.

American Society of Civil Engineers (ASCE) (2009). "2009 Report Card for America's Infrastructure", American Society of Civil Engineers, March 25, 2009 <http://www.asce.org/reportcard> (Dec. 15, 2010)

ATC-20 (1989), "Procedures for Postearthquake Safety Evaluations of Buildings", Report ATC-20, Applied Technology Council. Redwood City, CA.

ATC-20-2 (1995). "Addendum to the ATC-20 Post-earthquake Building Safety Evaluation Procedures." Applied Technology Council. Redwood City, CA.

ATC-35 (1999). "Earthquake Aftershocks - Entering Damaged Buildings, ATC-35 TechBrief 2", last visit: https://www.atcouncil.org/atc-pdf/atc35tb2.pdf (June 2008).

Bandera, A., Perez-Lorenzo, J., Bandera, J., and Sandoval, F., (2006) "Mean shift based clustering of Hough Domain for fast line segment detection", Pattern Recognition Letter, 27(2006): 578-586.

Bartel, J, (2001), "A picture of bridge health." NTIAC (Nondestructive Testing Information Analysis Center) Newsletter, 27~1, 1–4.

Bradski, G., and Kaehler, A., (2008), "Learning OpenCV: Computer Vision with the OpenCV", ISBN-13: 978-0596516130

Brilakis, I., Soibelman, L., and Shinagawa, Y., (2006) "Construction Site Image Retrieval Based on Material Cluster Recognition", Journal of Advanced Engineering Informatics, Elsevier Science, Volume 20, Issue 4, October 2006, Pages 443 – 452

Brilakis, I. and Soibelman, L., (2008) "Shape-Based Retrieval of Construction Site Photographs", Journal of Computing in Civil Engineering, American Society of Civil Engineers, Volume 22, Issue 1, January/February 2008, Pages 14 – 20

Brilakis, I., German, S., and Zhu, Z. (2011). "Visual Pattern Recognition Models for Remote Sensing of Civil Infrastructure.", Journal of Computing in Civil Engineering (in press)

Bureau of Transportation Statistics (BTS) (2008). "Condition of U.S. highway bridges: 1990-2008." <http://www.bts.gov> (Sept. 10, 2009)

Cantoni, V., Lombardi, L., Marco, P., and Sicard, N., (2001) "Vanishing point detection: representation analysis and new approaches", In: the Proceedings of the 11th International Conference on Image Analysis and Processing (ICIAP 2001), Pavia, Italy, pp. 90 - 94

CEE News, (2010), "Machine Vision-Based Damage Assessment", School of Civil and Environmental Engineering, Georgia Institute of Technology, <http://www.ce.gatech.edu/news/435/74/Machine-Vision-Based-Damage-Assessment> (July, 2010)

Chae, M.J., Iseley, T. and Abraham, D.M. (2003) "Computerized Sewer Pipe Condition Assessment" Proceedings of the ASCE International Conference on Pipeline Engineering and Construction, pp. 477-493, July 13-16, 2003, Baltimore, MD

Cheng, H., Shi, X. and Glazier, C., (2003). "Real-time image thresholding based on sample space reduction and interpolation approach'. Journal of Computing in Civil Engineering. 17(4): 264-272.

Cheok, G. S., Stone, W. C., Lipman, R. R., and Witzgall, C. (2000). "Ladars for construction assessment and update." Automation in Construction, 9(5–6): 463–477.

Chock, G., (2007). "ATC-20 Post-Earthquake Building Safety Evaluations Performed after the October 15, 2006 Hawaii Earthquakes Summary and Recommendations for Improvements (updated)." <http://www.scd.state.hi.us/HazMitPlan/chapter_6_appM.pdf> (Dec. 10, 2008)

Choi, N., Son, H., Kim, C., and Kim, H., (2008) "Rapid 3D Object Recognition for Automatic Project Progress Monitoring using a Stereo Vision System", In: the 25th International Symposium on Automation and Robotics in Construction, June 26-29, 2008, pp. 58 - 63

City of Lincoln (Lincoln), (2007), "Construction Inspector I/II", last visit: <http://www.ci.lincoln.ca.us/pagedownloads/Construction%20Inspector%20I-II%20web%2002-07.pdf> (June 12, 2007)

COMT (Council of Ministers) (2008). "Final Report Trends and Patterns in Skills and Labour Shortages." Council of Deputy Minister, Ottawa, Ontario <http://www.comt.ca/english/LabourSkills.pdf> (Feb. 4, 2011)

Cornelis, N. And Van-Gool, L., (2008) "Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware", In: Computer Vision and Pattern Recognition Workshop, 2008, IEEE Computer Society Conference, Anchorage, AK., pp. 1-8.

David, P. and DeMenthone, D., (2005) "Object Recognition in High Clutter Images using Line Features", In: Proceedings of the 10th IEEE International Conference on Computer Vision, Vol. 2, Issue 17-21, pp. 1581-1588.

Duda, R. O. and Hart, P. E. (1972). "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM, Vol. 15*, pp. 11–15

Emgu CV, (2010), "Emgu CV: OpenCV in .NET (C#, VB, C++ and More)"., last visit: http://www.emgu.com/wiki/index.php/Main_Page (Jun, 2010)

Engineering News Records (ENR). (2004). "Construction tops trillion dollar annual pace." Engineering News Records, 253(19), 23.

Federal Highway Administration (FWHA) . (1995). Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges. Washington, D.C.: U.S. Department of Transportation.

Federal Highway Administration (FHWA) (2001). "Reliability of Visual Bridge Inspection." Turner-Fairbank Highway Research Center, Federal Highway Administration, March, 2001. <http://www.tfhrc.gov/pubrds/marapr01/bridge.htm> (Dec. 12, 2007)

Federal Emergency Management Agency (FEMA), (2006), "National Urban Search and Rescue Response System – Structure Specialist Position Description", last visit: http://www.disasterengineer.org/library/Struct%20Spec%20PD%20July%202006.pdf (Dec. 2008)

Federal Emergency Management Agency (FEMA), (2009), "Module 1C Structural Engineering", Structural Collapse Technician Course – Student Manual, last visit: http://www.fema.gov/emergency/usr/sctc.shtm (Dec, 2009)

Felzenszwalb, P. and Huttenlocher, D., (2004), "Efficient Graph-based Image Segmentation", International Journal of Computer Vision, 59(2): 167 – 181.

Fernandes, L. and Oliveira, M., (2008) "Real-time line detection through an improved Hough transform voting scheme", Pattern Recognition, 41(1): 299–314.

FOX News, (2007). "Interview with Minn. Gov Pawlenty." <http://www.realclearpolitics.com/articles/2007/08/interview_with_minn_gov_pawlen.html > (Oct. 23, 2009)

Galietti, U. and Palumbo, D. (2010). "Application of thermal methods for characterization of steel welded joints." 14th International Conference on Experimental Mechanics, 6(38012).

Ge, F., Liu, T., Wang, S. and Stahl, J., (2008) "Template-Based Object Detection through Partial Shape Matching and Boundary Verification", International Journal of Signal Processing, Vol. 4, No. 1, pp. 148 – 157

Golparvar-Fard M., Peña-Mora F., and Savarese S. (2009). "D4AR- A 4-Dimensional augmented reality model for automating construction progress data collection, processing and communication." Journal of Information Technology in Construction (ITcon), Special Issue Next Generation Construction IT: Technology Foresight, Future Studies, Roadmapping, and Scenario Planning, 14, 129-153, http://www.itcon.org/2009/13.

Gong, J. and Caldas, C., (2009), "Construction Site Vision Workbench: A Software Framework for Real-Time Process Analysis of Cyclic Construction Operations", In: Proceedings of the 2009 ASCE International Workshop on Computing in Civil Engineering, Austin, TX.

Gong, J. and Caldas, C., (2010). "Computer Vision-Based Video Interpretation Model for Automated Productivity Analysis of Construction Operations." Journal of Computing in Civil Engineering, 24(3): 252 – 263.

Gordon, C., Boukamp, F., Huber, D., Latimer, E., Park, K., Akinci, B. (2003) "Combining Reality Capture Technologies for Construction Defect Detection: A

Case Study", In: 9th EuropIA International Conference (EIA9): E-Activities and Intelligent Support in Design and the Built Environment, Oct. 2003, Istanbul, Turkey, pg. 99-108.

Green, B., (2002), "Edge detection tutorial", <http://www.pages.drexel.edu/~weg22/edge.html> (Dec. 21, 2008)

Gucunski, N. and Consolazio, G. (2006) "Concrete bridge deck delamination detection by integrated ultrasonic methods." International Journal of Materials and Product Technology, Vol. 26, Number 1 – 2, pp. 19 – 34.

Guo, W., Soibelman, L. and Garrett, J (2008).. "Imagery Enhancement and Interpretation for Remote Visual Inspection of Aging Civil Infrastructure," ICCCBE-XII & INCITE 2008, October 16-18, 2008, Tsinghua University, Beijing.

Guo, W., Soibelman, L., and Gareett, J., (2009) "Automated defect detection for sewer pipeline inspection and condition assessment", Automation in Construction, 18(5): 587-596

Guru, D, Shekar, B. and Nagabhushan, P., (2004) "A simple and robust line detection algorithm based on small eignevalue analysis", Pattern Recognition Letters, 25(2004): 1-13

Hsu C-W, Chang, C-C., and Lin, C-J, "A Practical Guide to Support Vector Classification", <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf> (May. 05, 2010).

Ikeuchi, K. and Sato, Y., (2001), "Modeling from Reality", In: the Proceedings of 3-D Digital Imaging and Modeling, Quebec, Canada, 117-126

Iyer, S. and Sinha, S.K. (2006). "Segmentation of pipe for crack detection in buried sewers." Computer-Aided Civil and Infrastructure Engineering, 2006, 21: 395–410.

Lee, S., Chang L.M. and Chen, P.H., (2005), "Performance comparison of bridge coating defect recognition methods", Corrosion 61 (2005) (1), pp. 12–20.

Lee, S., Chang L.M., and Skibniewski, M., (2006), "Automated recognition of surface defects using digital color image processing", Automation in Construction, Vol 15, Issue 4, pp. 540-549

Li, J., “An empirical comparison between SVMs and ANNs for Speech Recognition”, <http://www.cs.rutgers.edu/~mlittman/courses/ml03/iCML03/papers/li.pdf> (April. 05, 2010).

Liu, Y., Ikenaga, T., and Goto, S., (2007) “An MRF model-based approach to the detection of rectangular shape objects in color images”, J. of Sig. Processing 87(2007): 2649-2658

Liu, Z., Shahrel, A., Ohashi, T., and Toshiaki, E., (2002), “Tunnel crack detection and classification system based on image processing”, In: Proc. SPIE Vol. 4664, p. 145-152, Machine Vision Applications in Industrial Inspection X.

Lukins, T. C. and E. Trucco (2007). “Towards Automated Visual Assessment of Progress in Construction Projects.” Proceedings of the British Machine Vision Conference, Warwick, UK. pp. 142-151.

Jaselskis, E.J., Gao, Z. and Walters, R.C. (2005). “Improving transportation projects using laser scanning.” J. Constr. Eng. Manage., 131(3), 377-384.

Jauregui, D., and White, K. (2003). Implementation of Virtual Reality in Routine Bridge Inspection. Journal of Transportation Research Record, 1827, 29-35

Johnson, K., (2004). “San Simeon Earthquake, City of Paso Robles Emergency Response Report”, last visit: http://www.prcity.com/government/pdf/EQResponseRpt.pdf (March 2009).

Jung, C. and Schramm, R., (2004), “Rectangle detection based on a windowed Hough transform”, In: Proceedings of Computer Graphics and Image Processing, 2004. 17th Brazilian Symposium on 17-20 Oct. 2004, pp: 113 - 120

Kamat, V. R., and El-Tawil, S. (2007). "Evaluation of Augmented Reality for Rapid Assessment of Earthquake-Induced Building Damage", Journal of Computing in Civil Engineering, Vol. 21, No. 5, American Society of Civil Engineers, Reston, VA, 303-310.

Kamoi, A., Okamoto, Y., and Vavilov, V. (2004). “Study on Detection Limit of Buried Defects in Concrete Structures by Using Infrared Thermography.” Key Engineering Materials, Vol. 270 – 273, pp. 1549-1555

Ke, Y. and Sukthankar, R., (2004) "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors", Computer Vision and Pattern Recognition, (CVPR), 2004, Volume 2, Issue , 27 June-2 July 2004 Page(s): II-506 - II-513.

Kirk, R. and Mallett, W., (2007), "Highway bridges: conditions and the federal/state role", CRS report for congress, < http://www.fas.org/sgp/crs/homesec/RL34127.pdf> (April, 2010)

Koglin, T., (2007), "Movable Bridge Engineering." John Wiley&Sons, Inc. ISBN: 978-0-471-41960-0

Kosovac, B., Froese, T. and Vanier, D., "Integrating Heterogeneous Data Representations in Model-Based AEC/FM Systems", Proceedings of Construction Information Technology 2000 – CIB-W78, IABSE, EG-SEA-AI International Conference on Information Technology, Reykjavik, Iceland, 28-30 June, 2000, G. Gudnason (Ed.), Icelandic Building Research Institute, Vol. 2, pp. 556-567

Kosecka, J., and Zhang, W., (2002), "Video compass", In: European Conference on Computer Vision, pages 657 – 673, 2002.

Kostoulas, D., Aldunate, R., Peña-Mora, F. and Lakhera, S., (2006), "A Decentralized Trust Model to Reduce Information Unreliability in Complex Disaster Relief Operations", In Proc. 13th EG-ICE Workshop on Intelligent Computing in Engineering and Architecture, Ascona, Switzerland, June 25-30, 2006, LNCS 4200, pp. 383-407

Kotsiantis, S. (2007), "Supervised Machine Learning: A Review of Classification Techniques", Informatica 31(2007): 249 - 268

Kottapalli, V. A., Kiremidjian, A. S., Lynch, J. P., Carryer, E., Kenny, T. W., Law, K. H., and Lei, Y. (2003). "Two-tiered wireless sensor network architecture for structural health monitoring." Proceedings of SPIE - The International Society for Optical Engineering - Smart Structures and Materials, 5057, 8-19.

Kwak, N. and Oh, J. (2009). " Feature extraction for one-class classification problems: Enhancements to biased discriminant analysis." *Pattern Recogn.* 42(1): 17-26

Kwan, M. and Lee, J., (2005), "Emergency response after 9/11: the potential of real-time 3D GIS for quick emergency response in micro-spatial environments", Comput., Environ., and Urban Systems, vol. 29, pp. 93-113.

Lankton, S. and Tannenbaum, A., "Localizing region-based active contours", IEEE Transactions on Image Processing, 17(11): 2029 – 2039

Larsen, C. (2010). "3D Reconstruciton of Buildings from Images with Automatic Façade Refinement." Master's Thesis in Vision, Graphics and Interactive Systems, Aalborg University, <http://projekter.aau.dk/projekter/files/32366963/report.pdf> (Jan. 25, 2010)

Lazebnik, S. Schmid, C. and Ponce, J., (2004) "Semi-Local Affine Parts for Object Recognition," In: Proceedings of the British Machine Vision Conference, Kingston, UK, September 2004, vol. 2, pp. 959-968.

Lee Y., Koo, H., and Jeong, C., (2006a) "A straight line detection using principle component analysis", Pattern Recognition Letter, 27(2006): 1744 -1754.

Lee, S., Chang L.M., and Skibniewski, M., (2006b), "Automated recognition of surface defects using digital color image processing", Automation in Construction, Vol 15, Issue 4, pp. 540-549

Leung, T., and Malik, J., (2001), "Representing and Recognizing the Visual Appearance of Matrials using Three-dimensional Textons", International Journal of Computer Vision, 43(1): 29 – 44.

Lowe, D. (2004). "Distinctive image features from scale-invariant." International Journal of Computer Vision, 60(2): 91 – 110.

Lin, Z., Kim, S., Lee, W., and Kweon, I., (2004) "Robust Invariant Feature Extraction for Object Recognition and Natural Landmark based Autonomous Navigation", In: The 1st International Conference on Ubiquitous Robots and Ambient Intelligence, COEX, Seoul, Korea, 1-3 December, 2004. pp.78-84

Liu, T., Moore, A. Gray, A., and Yang, K. (2004). "An investigation of practical approximate nearest neighbor algorithms." In Proceedings of Neural Information Processing Systems, Vancouver, BC, Canada, pp. 825 – 832.

Ma, Z., Shen, Q. and Zhang, J. (2005), "Application of 4D for dynamic site layout and management of construction projects", Automation in Construction, 2005, 14(3) pp: 369-381

MDOT (2007). “POINTIS bridge inspection manual.” Michigan Department of Transportation. <http://www.michigan.gov/documents/mdot/MDOT_PontisManual_2007_195365_7.pdf > (Dec. 12, 2010)

Mehta, S. and Pena-Mora, F., (2009), “Expediting Assessment of Damaged Buildings during Disaster Response using Mobile Ad-hoc Networks”, In: Proceedings of the ASCE International Workshop on Computing in Civil Engineering, Austin, Texas, June 24-27, pp. 217-226.

Micusik, B. Wildenauer, H. Kosecka, J., (2008) “Detection and matching of rectilinear structures”, In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp. 1-7, Anchorage, AK.

Mikolajczyk, K. and Schmid, C., (2005) “A performance evaluation of local descriptors”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(10): 1615-1630.

MnDOT (2007). “Mn/DOT bridge inspection report.”, Minnesota State Department of Transportation, 2007. <http://www.dot.state.mn.us/i35wbridge/pdfs/bridge_inspection_report_06-15-06.pdf > (Jan. 20, 2010)

Momma (2008). “OpenCV Thinning”, < http://www.eml.ele.cst.nihon-u.ac.jp/~momma/wiki/wiki.cgi/OpenCV/> (March 1, 2011)

Moore, M., Phares, B., Graybeal, B., Rolander, D., & Washer, G. A. (2001). “Reliability of Visual Inspection for Highway Bridges, Volume I: Final Report and, Volume II: Appendices.” FHWA-RD-01-020(021), U.S. Department Of Transportation, Washington, D.C.

Muja, M. and Lowe, D., (2009) "Fast approximate nearest neighbors with automatic algorithm configuration," In: International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal (Feb 2009), pp. 129 - 139.

Naeim, F., Hagie, S., Alimoradi, A. and Miranda, E., (2006), “Automated Post-Earthquake Damage Assessment and Safety Evaluation of Instrumented Buildings”, In: Advances in Earthquake Engineering for Urban Risk Reduction, Vol. 66, pp. 117-134.

NCHRP (National Cooperative Highway Research Program). (2004). "Automated Pavement Distress Collection Techniques." Transportation Research Record, NCHRP Synthesis 334, 2004.

NASA, (2009). Disaster Assistance and Rescue Team—Structural Assessment Team, last visit: http://dart.arc.nasa.gov/SAT/SAT.html (Dec, 2009)

National Academy of Engineering (NAE) (2008). "Grand Challenges for Engineering Restore and improve urban infrastructure", National Academy of Engineering, Washington D.C. <http://www.engineeringchallenges.org/cms/8996/9136.aspx> (Dec. 10, 2010)

National Institute of Standards and Technologies (NIST), (2005), "Federal Building and Fire Safety Investigation of the World Trade Center Disaster: Final Report of the National Construction Safety Team on the Collapses of the World Trade Center Tower (DRAFT)", NIST, U.S.

Neto, J. and Arditi, D., (2002) "Using Colors to Detect Structural Components in Digital Pictures", Computer Aided Civil and Infrastructure Engineering, 17(2002): 61-76

Nikolaos, C., Anagnostopoulos, E., Anagnostopoulos, I., Loumos, V. and Kayafas, E., (2006) "A license plate-recognition algorithm for intelligent transportation system applications, IEEE Transactions on Intelligent Transportation Systems, 7(3): 377-392.

Nister, D. and Stewenius, H., (2006) "Scalable recognition with a vocabulary tree", In: Computer Vision and Pattern Recognition, New York, NY., Vol. 2: pp. 2161–2168.

NJDOT (2009), "Bridge Inspection Work Zone, Setup Guide", <http://www.state.nj.us/transportation/eng/structeval/pdf/InspectionWorkZoneSetup.pdf > (Sept. 19, 2009)

Reality Measurements Inc. (2009). < www.realitymeasurements.com> (Dec. 12, 2009)

Park, M.W., Makhmalbaf, A. and Brilakis, I. (2011) "Comparative Study of Vision Tracking Methods for Tracking of Construction Site Resources", Journal of Automation in Construction, Elsevier, (in press).

Patterson A.M., Dowling G.R., and Chamberlain D.A., (1997), "Building Inspection: Can Computer Vision Help", Automation in Construction, 7(1): 12 – 20.

Phares, B., Washer, G., Rolander, D., Graybeal, B., & Moore, M., (2004), "Routine highway bridge inspection condition documentation accuracy and reliability." J. of Bridge Engr. 9(4): 403-413.

Potmesil, M. and Chakravarty, I. (1982), "Synthetic Image Generation with a Lens and Aperture Camera Model", ACM Transactions on Graphics, 1, ACM, pp. 85–108,

Prine, D. W., (1995), "Problems Associated with Nondestructive Assessment of Bridges", SPIE Conference on Nondestructive Assessment of Aging Bridges and Highways, Oakland, CA, June 6-7, 1995

Ratsch, G., Mika, S., Schölkopf, B., and Müller, K. 2002. "Constructing Boosting Algorithms from SVMs: An Application to One-Class Classification". *IEEE Trans. Pattern Anal. Mach. Intell.* 24(9): 1184 – 1199

Renois, C. (2010). "Haitians Angry over Slow Aid." <http://www.theage.com.au/world/haitians-angry-over-slow-aid-20100204-ng2g.html > (April, 23, 2010)

Remondino F. and El-Hakim, S., (2006), "Image-based 3D Modeling: A Review", The Photogrammetric Record, 2006, 21(115): 269-291.

Rother, C., (2000), "A new Approach for Vanishing Point Detection in Architectural Environment", In Proc. 11th British Machine Vision Conference, Bristol, UK, 11-14 September 2000

Schmid, C. (2001), "Constructing models for content-based image retrieval." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 39-45, 2001.

Scordino, C., (2006) "Object Recognition using Local Invariant Features", Seminar Talk about "Language and Intelligence", University of Pisa, <http://medialab.di.unipi.it/web/Language+Intelligence/objectrecognition.ppt> (Dec. 10, 2008)

Shaw, D. and Barnes, N., (2006) “Perspective Rectangle Detection”, In: Applications of Computer Vision, Workshop at the European Conference on Computer Vision (ECCV), May, 2006.

Silpa-Anan, C. and Hartley, R., (2008) “Optimized KD-trees for fast image descriptor matching”, In: Computer Vision and Patter Recognition, Dec. 23-28, 2008, Anchorage, AK, pp. 1-8.

Sitar M., (2005). “A maintenance strategy based on prevention, inspection and detection keeps facilities operating safely and cost-effectively.” Maintenance Solutions, <http://www.facilitiesnet.com/ms/article.asp?id=3455&keywords=concrete,%20flooring,%20floor%20maintenance,%20concrete%20maintenance> (June, 10, 2007)

Sinha, S., & Fieguth, P. (2006). Automated detection of cracks in buried concrete pipe images. Automation In Construction , 15 (1), 58-72.

Sternberg, H., Kersten, T. and Kinzel, R. (2004). "Terrestrial 3D laser scanning - Data acquisition and object modeling for industrial as-built documentation and architectural applications." ISPRS ThS-17 Laser Scanning Acquisition and Modeling technologies, Istanbul, Turkey.

Sunkpho, J. (2001). A Framework for Developing Field Inspection Support Systems. Ph.D Disseration, Department of Civil and Environmental Engineering, Carnegie Mellon University.

Suwwanakarn, S., Zhu, Z. and Brilakis, I. (2007) "Automated Air Pockets Detection for Architectural Concrete Inspection", ASCE Construction Research Congress, 6-8 May 2007, Freeport, Bahamas

Tan A., and Gilbert, D. (2003), “An empirical comparison of supervised machine learning techniques in bioinformatics”, In: Proceedings of the First Asia-Pacific Bioinformatics Conference on Bioinformatics 2003 - Volume 19 (Adelaide, Australia).

Tax, D.. (2001), “One-class classification”, Ph.D thesis, Delft Tech. University

Thompson, P. and Shepard, R. (2000). “AASHTO Commonly Recognized Bridge Elements, Successful Applications and Lessons Learned National Workshop on Commonly Recognized Measures for Maintenance.” <http://www.pdth.com/images/coreelem.pdf> (Sept. 21, 2009)

Transportation Pooled Fund Program (TPFP), (2009), "TPF Studies: Support of the Transportation Curriculum Coordination Council", Federal Highway Agencies, last visit: http://www.pooledfund.org/projectdetails.asp?id=435&status=4, (Sept. 3, 2009)

Teizer, J., Haas, C., and Caldas, C., and Bosche, F., (2007) "Real-Time Three-Dimensional Occupancy Grid Modeling for the Detection and Tracking of Construction Resources", Journal of Construction Engineering and Management, 133(11): 880-888

Tsai, M-H, Liu, L., and Pena-Mora,F., (2007), "A building blackbox for urban disaster response and relief", In: Proceedings of the ASCE International Workshop on Computing in Civil Engineering, Pittsburg, Pennsylvania, July 24-27, pp. 809-816.

United Kingdom Fire Service Search and Rescue Team (UKFSSART), (2007), "Structure Collapse - A Guide for Emergency Personnel", last visit: <http://www.ukfssart.org.uk/index.htm> (Dec. 11, 2009)

USA Today (2008). "Report: Money woes may have led to bridge collapse." <http://www.usatoday.com/news/nation/2008-05-21-bridgecollapse-report_N.htm> (Sept. 12, 2010)

Varma, M. and Zisserman, A., (2005), "A statistical approach to texture classification from single images", International Journal of Computer Vision, Vol. 62, Issue 1-2, pp. 61-81

Wasserman, E. (2007). "Are Tennessee Bridges Safe?" <http://www.tdot.state.tn.us/documents/Wasserman_editorial.pdf> (Sep. 21, 2009)

Wickens, T., (2002) "Elementary Signal Detection Theory", New York Oxford University Press, ISBN 0195092503

Wu, Y., and Kim, H., (2004) "Digital Imaging in Assessment of Construction Project Progress", In: the 21th International Symposium on Automation and Robotics in Construction, pp. 537 – 542.

Yamaguchi, T. and Hashimoto, S., (2009). "Fast crack detection method for large-size concrete surface images using percolation-based image processing". Machine Vision and Applications, 11(5): 797-809.

Yu, S.-N., Jang, J.-H., & Han, C.-S. (2007). Auto Inspection System Using a Mobile Robot for Detecting Concrete Cracks in a Tunnel. Automation in Construction, 16 (3), 255-261.

Zhu, Z. and Brilakis, I. (2009). "Comparison of civil infrastructure optical-based spatial data acquisition techniques.", J. Comput. Civ. Eng., 23(3), 170-177.

VITA

ZHENHUA ZHU

Zhenhua Zhu was born in Hangzhou, China. He holds a B.E. in civil engineering and an M.E. in computer science and technology. In 2006, He started to pursue a doctorate in civil engineering with a specialization in construction engineering and management. Zhenhua Zhu is an active member in several academic and professional organizations, and officially serves as a reviewer for ASCE journals. His research interests include pattern recognition and filtering techniques for structural element recognition, and defects and damage detection and evaluation for civil infrastructure assessment and rehabilitation. In 2009, he received the Best Paper Award of the ASCE Construction Research Congress.